Design Compiler®
Tutorial Using Design Vision™

Version B-2008.09, June 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (^{SM})

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

Preface

This preface includes the following sections:

What's New in This Release

Information about new features, enhancements, and changes, along with known problems and limitations and resolved Synopsys Technical Action Requests (STARs), is available in the *Design Compiler Release Notes* in SolvNet.

To see the *Design Compiler Release Notes,*

1. Go to the release notes page on SolvNet located at the following address:

   https://solvnet.synopsys.com/ReleaseNotes

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.
2. Select Design Compiler, then select a release in the list that appears at the bottom.

About This Tutorial

This tutorial shows how you use Design Compiler with the Design Vision graphical interface to optimize, analyze, and correct a sample design. Through tutorial exercises, you will learn the basics of Design Compiler synthesis and the principal features of Design Vision at the same time.

Audience

This tutorial is intended for engineers who are familiar with ASIC design but are not familiar with Design Compiler or Design Vision. A working knowledge of high-level design techniques, a hardware description language such as VHDL or Verilog, the operating system for your computer, and various commands derived from the UNIX operating system are assumed.

Related Publications

For additional information about Design Compiler, see Documentation on the Web, which is available through SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

You might also want to refer to the documentation for the following related Synopsys products:

- Automated Chip Synthesis
- Design Budgeting
- Design Vision
- DesignWare components
- DFT Compiler
- PrimeTime
- Power Compiler
- HDL Compiler

Also see the following related documents:

- *Using Tcl With Synopsys Tools*

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
Courier bold	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1`* [*`pin2 ... pinN`*]
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to https://solvnet.synopsys.com (Synopsys user name and password required), and then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

Introduction to the Design Compiler Tutorial

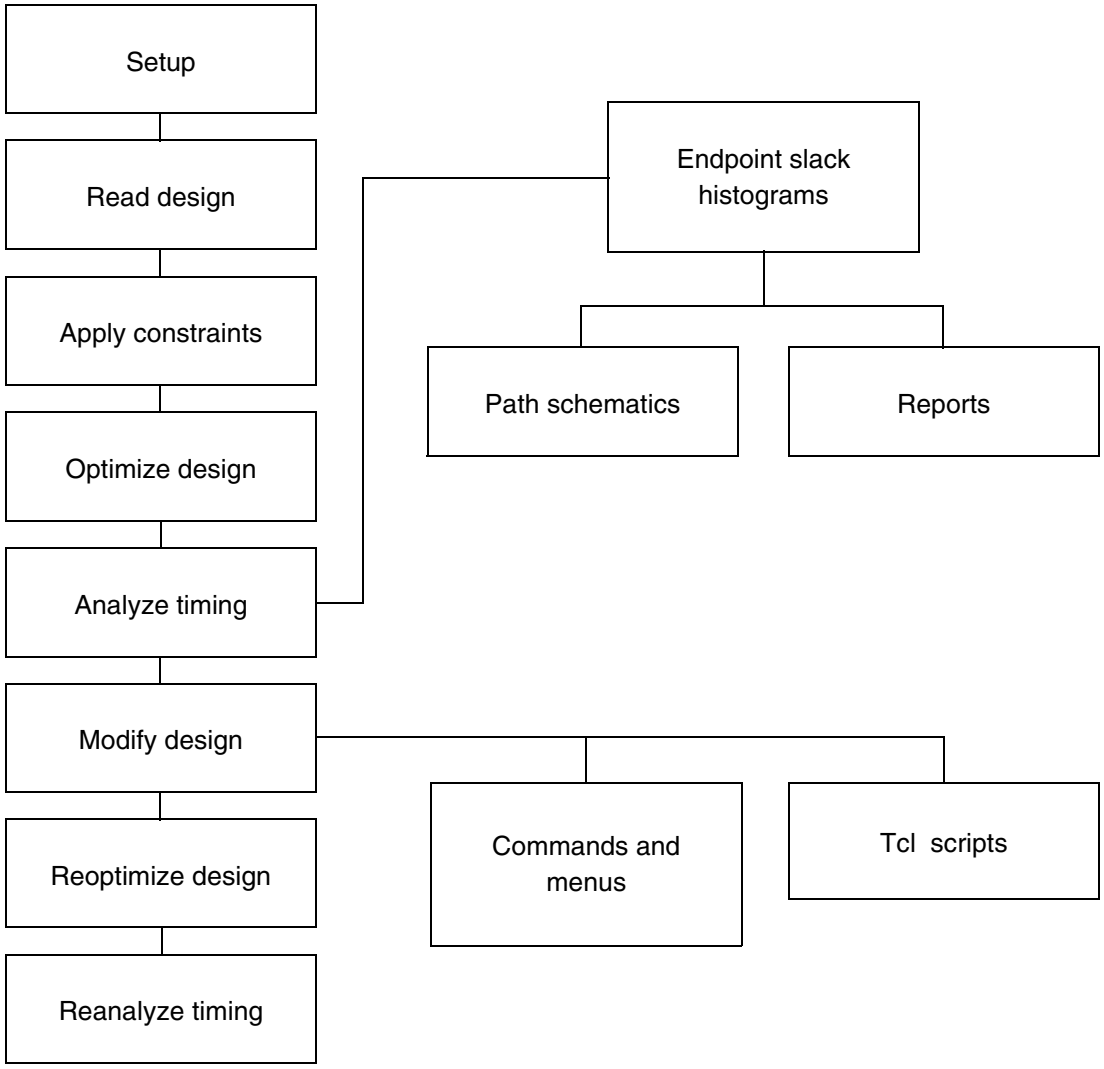
This chapter provides background information only. You will not do any tutorial exercises in this chapter. Exercises begin in Chapter 2.

This chapter includes the following sections:

Basic Logic Synthesis Process

Figure 1-1 shows the design flow you will follow in this tutorial, using the Design Vision graphical user interface (GUI).

Figure 1-1 Tutorial Design Flow



This flow is similar to the basic design flow you would use for a real design application. You will learn how to carry out the steps of this flow in Chapters 2, 3, 4, and 5.

License Requirements

To run this tutorial, you need the following licenses:

- Design-Compiler
- Design-Vision

Also, if you want to use any of the Design Compiler advanced functionality, you must check out the appropriate licenses.

Synopsys licensing software and the documentation describing it are now separate from the tools that use it. You install, configure, and use a single copy of Synopsys Common Licensing (SCL) software for all Synopsys tools. Because it provides a single, common licensing base for all Synopsys tools, SCL reduces licensing administration complexity, minimizing the effort you expend in installing, maintaining, and managing licensing software for Synopsys tools.

For complete Synopsys licensing information, see the following documents:

- *Common Licensing Quick Start*

  This booklet provides instructions on how to obtain an electronic copy of your license key file and how to install and configure SCL.

- *Common Licensing Installation and Administration Guide*

  This guide provides detailed information on SCL installation and configuration, including examples of license key files and troubleshooting guidelines.

Supported Platforms

Design Vision is supported on the same platforms that support Design Compiler and the other synthesis tools. For details, see the *Installation Guide*.

Your hardware and operating system vendor has required patches available for your system. For more information about supported platforms and the operating system patches necessary to run Synopsys software on supported platforms, use your browser to go to

http://www.synopsys.com/Support/Licensing/Pages

From this Web page you can navigate to the Supported Platforms Guide page for your release.

Input and Output File Formats

Design Vision gives you access to all the files supported by Design Compiler. See Chapter 1 of the *Design Vision User Guide* for a list of these formats. The tutorial exercises use only the file formats listed in Table 1-1.

Table 1-1 File Formats Used in the Tutorial Exercises

Data	Formats
Netlist	VHDL
Netlist	Synopsys database format (.ddc)
Command Script	Tool command language (Tcl)
Library	Synopsys internal library format (.lib)
Library	Synopsys database format (.db)
Symbol Library	Synopsys database format (.sdb)

Preparing to Run the Tutorial

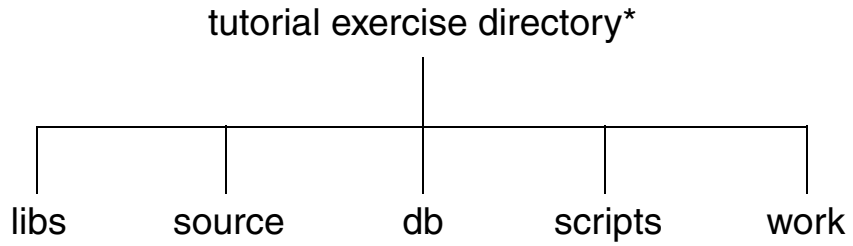
Design Compiler and Design Vision must be installed before you can run the tutorial. The tutorial directories and files are installed under the dv_tutorial directory in the Synopsys root path during standard installation. Copy this directory to a working directory, preferably with the same directory name, by entering the UNIX command

```
% cp -r $SYNOPSYS/doc/syn/dv_tutorial .
```

The dv_tutorial directory contains the three subdirectories risc_design, risc_design-flse-path, and risc_design-mult-clock. These subdirectories contain the files you need to run the tutorial exercises presented in the following chapters.

Figure 1-2 shows the general directory structure for the risc_design, risc_design-flse-path, and risc_design-mult-clk subdirectories.

Figure 1-2 Tutorial Directory Structure



* tutorial exercise directory is one of the following directories:

- dv_tutorial/risc_design or
- dv_tutorial/risc_design-flse-path or
- dv_tutorial/risc_design-mult-clk

risc_design Directory

The risc_design directory is used for the main tutorial exercise (Chapters 2, 3, and 4). The directory consists of its own .synopsys_dc.setup file and the following subdirectories and files:

- source directory

  This directory contains the VHDL design source files. Make sure the directory contains the following files:

  - ALU.vhd
  - CONTROL.vhd
  - DATA_PATH.vhd
  - INSTRN_LAT.vhd
  - PRGRM_CNT.vhd
  - PRGRM_CNT_TOP.vhd
  - PRGRM_DECODE.vhd
  - PRGRM_FSM.vhd
  - REG_FILE.vhd

  - RISCTYPES.vhd
  - RISC_CORE.vhd
  - STACK_FSM.vhd
  - STACK_MEM.vhd
  - STACK_TOP.vhd
- libs

  This directory contains the technology library used to map the risc_core design. Make sure the directory contains the following files:

  - core.sdb
  - core_typ.db
- scripts

  This directory contains the runtime script and the top-level constraints file. Make sure the directory contains the following files:

  - run.tcl
  - top-level.tcl
- db

  This directory is empty when you start the tutorial exercise. As you carry out the exercise, you will generate and store the mapped and unmapped .ddc design files in this directory.
- work

  This directory is the work library and is used by Design Compiler to store intermediate design file results, obtained as you run the tutorial exercise.

risc_design-flse-path Directory

The risc_design-flse-path directory is used for the false path tutorial exercise (Chapter 5). The directory consists of its own .synopsys_dc.setup file and has the same directory structure as the risc_design directory.

This exercise uses the same technology library as the main exercise, that is, the contents of the libs directory are the same. Also, the db and work directories are used the same way in this exercise as in the main exercise. The source and scripts directories are different.

- source directory

  Make sure this directory contains its own STACK_FSM.vhd design file and no other source files. Be careful not to overwrite or replace this file with a file having the same name from a different source directory.

- scripts directory

  Make sure this directory contains the constraints file fsm_flse_path_constraints.tcl.

risc_design-mult-clk Directory

The risc_design-mult-clk directory is used for the multiple clock tutorial exercise (Chapter 5). The directory consists of its own .synopsys_dc.setup file and has the same directory structure as the risc_design directory.

This exercise uses the same technology library as the main exercise, that is, the contents of the libs directory are the same. Also, the db and work directories are used the same way in this exercise as in the main exercise. The source and scripts directories are different.

- source directory

  Make sure this directory contains its own STACK_FSM.vhd design file and no other source files. Be careful not to overwrite or replace this file with a file having the same name from a different source directory.

- scripts directory

  Make sure this directory contains the constraints file fsm_mult_clk_constraints.tcl.

Starting and Exiting Design Compiler From Design Vision

You start Design Vision from a UNIX or Linux shell in the *appropriate* directory. Each of the three tutorial exercises has its own directory, namely, risc_design, risc_design-flse-path, and risc_design-mult-clk. These directories have their own .synopsys_dc.setup file that must be used. Therefore, before invoking Design Vision for a particular exercise, make sure you change to the correct directory.

To start Design Vision, enter

```
% design_vision
```

You can also start Design Vision from a dc_shell session. To start Design Vision, use the following command at the dc_shell prompt:

```
dc_shell> gui_start
```

The `-gui` option invokes the GUI automatically from dc_shell.

```
% dc_shell -gui
```

The Design Vision window, shown in Figure 1-3 on page 1-9, is displayed. In addition, the design_vision prompt appears in the shell where you started Design Vision and you are ready to begin the tutorial exercise. In this tutorial, all scripts are Tcl scripts. You use the dctcl command language to interact with Design Vision.

You can exit from Design Vision while still keeping the original dc_shell session active in the terminal window, use the `gui_stop` command at the dc_shell prompt.

You can also use, Choose File > Exit GUI from the menu bar and click OK in the warning message box, or enter the `exit` command on the command line to exit from Design Vision.

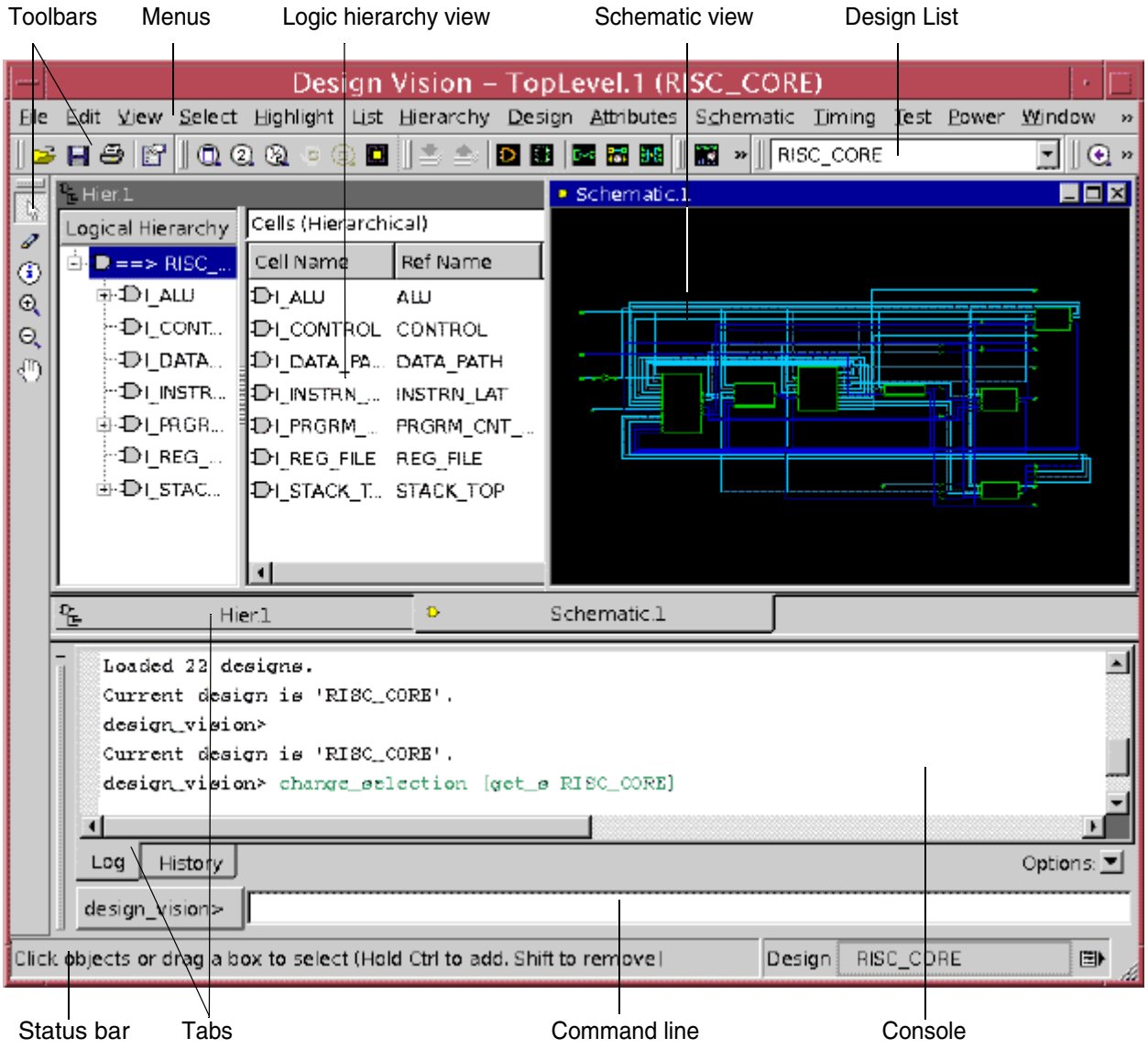
Note:

Design Compiler does not automatically save the designs that you work on. To save your designs, use the File > Save or File > Save As menu commands, or enter the `write -format -ddc` command on the command line.

Exploring the Design Vision Interface

The Design Vision window appears by default when you start Design Vision. Figure 1-3 shows what the Design Vision window looks like after you read in a design and open a design schematic for the top-level design.

Figure 1-3 Design Vision Window



The Design Vision window consists of a title bar, a menu bar, and several toolbars at the top of the window and a status bar at the bottom of the window. The title bar and menu bar are always visible. You can display or hide individual toolbars or the status bar.

You use the workspace area between the toolbars and the status bar to display view windows and panels. When you start a Design Vision session, a logic hierarchy view (sometimes called the “hierarchy browser”) and the console appear in the workspace. The analysis tasks that you perform during the session determine which other view windows and panels you open.

- Logic hierarchy view

  The Logical hierarchy view helps you explore your design and gather information. The view is divided into the following two panes:

  - Instance tree, on the left
  - Objects table, on the right

  The instance tree lets you quickly navigate the design hierarchy and see the relationships among its levels. Each instance is an occurrence of a design loaded in Design Compiler memory. An instance is also known as a cell. If you select a hierarchical instance (an instance that contains other instances), information about that instance appears in the objects table. You can Shift-click or Control-click instance names to select combinations of instances.

  By default, the objects table displays information about instances belonging to the selected instance in the instance tree. To display information about other types of objects, select the object types in the list above the objects table. You can display information about hierarchical cells, all cells, pins and ports, pins of child cells, and nets.

  If some of the text in a column is missing because the column is too narrow, you can hold the pointer over it to display the information in an InfoTip. You can also adjust the width of a column by dragging its right edge left or right.

- Console

  The console provides a command-line interface and displays information about the commands you use during the session in the following three views:

  - Log view
  - History view

  You can enter dctcl commands on the command line at the bottom of the console. Enter these commands just as you would enter them at the dctcl prompt in a standard UNIX or Linux shell. Design Vision echoes the dctcl command output (including processing messages and any warnings or error messages) in the console log view.

  The log view provides the session transcript. The history view provides a list of the commands that you have used during the session.

  To select a view, click the tab at the bottom of the console. The log view is displayed by default when you start Design Vision.

In addition to the logic hierarchy view, you can display design information in the following views:

- Timing analysis driver
- Path Inspector windows
- Histogram views (endpoint slack, path slack, and net capacitance)
- Path profile views
- Schematic views (design schematics, path schematics, and symbol views)
- Report views
- List views (cells, ports, pins, nets, and designs)

Note:
Unlike other view windows, path inspector windows contain their own menus and toolbars, and they are not confined within the workspace in the Design Vision window.

Design Vision displays a tab at the bottom of the View Area for each undocked view you open.

You can adjust the sizes of view windows for viewing purposes, and you can move them to different locations within the Design Vision window. When you click anywhere within a view window, Design Vision highlights its title bar to indicate that it has the focus (that is, it is the active view) and can receive keyboard and mouse input. For more details, see the “View Windows” topic in online Help.

To learn more about the functions of menu commands and toolbar buttons, see the “Menu Bar” and “Toolbars” topics in online Help.

The status bar, at the bottom of the window, displays current information about the session, such as the number and type of selected objects. If you hold the pointer over a menu command, a toolbar button, or a tab in the workspace, the status bar displays a brief message about the action that the command, button, or tab performs. To quickly display the list of selected objects in the Selection List dialog box, you can click the button at the right end of the status bar.

You can open additional Design Vision windows and use them to compare views, or different design information within a view, side by side. All open Design Vision windows share the same designs in memory, the same current timing information, and the same open views in the View Area. However, you can configure the views independently for each window. To learn more, see the “Opening New Design Vision Windows” topic in online Help.

You will learn how to use the Design Vision menus and their commands in the tutorial exercises that follow. For more information about the GUI and how to use it, see the *Design Vision User Guide* and the following online Help topics:

- Design Vision window
- Menu Bar
- Toolbars
- View Windows
- Status Bar
- Hierarchy Browser
- Command Console

Note:
To access the Help system from the Help menu in the main Design Vision window, choose Help > Online Help.

Take some time now to familiarize yourself with the Design Vision window, especially the menus, and with the online Help system. Later, after you read designs into memory, you can further explore the menu commands, the toolbar functions, the logical hierarchy view, and the console.

Getting Help in Design Vision

You can get help in Design Vision in two ways: from the online Help system and the man pages. These help systems provide a level of detailed information beyond that found in the tutorial exercises.

As you carry out the tutorial exercises, use online Help and the man pages whenever you want more information about the interface and how to use it, or more information about Design Compiler commands.

Design Vision Online Help

The Design Vision Online Help is available in the Design Vision GUI. You access the Help system from the Help menu in the main Design Vision window.

Design Vision Online Help explains the details of tasks that you can perform. For example, if you need help performing a step in a procedure presented in a tutorial exercise, you can find the information you need in the online Help system.

Information in online Help is grouped in the following categories:

- Feature topics

  Overviews of Design Vision window components and tools.

- How to topics

  Procedures for accomplishing synthesis and analysis tasks.

- Reference topics

  Explanations of views, menus, toolbars and panels, and some dialog boxes

To access online Help in Design Vision,

1. Choose Help > Online Help.

   The online Help system appears in the browser window and displays the Welcome topic for Design Vision Help.

2. Use the navigation frame (leftmost frame) to find the information you need in one of the following ways:

   - Find topics in the hierarchical organization of the Help system by clicking Contents and expanding the appropriate books until you find the information you need.
   - Find topics about a subject by clicking Index and looking for the subject in the alphabetical listing.
   - Search for key words found in topics by clicking Search and entering the keywords.

     If more than one topic has the words you are searching for, you must select the appropriate topic from a list of topics.

Design Vision online Help is a browser-based HTML Help system designed for viewing in the Firefox and Mozilla Web browsers. For details on setting up your browser for optimal Help viewing, see the *Design Vision User Guide*.

Note:
Before you can access online Help from within Design Vision, the Web browser executable file must be listed in your UNIX or Linux path variable.

Design Vision online Help makes extensive use of JavaScript and cascading style sheets (CSS). If your browser encounters problems displaying online Help, open the browser preferences and make sure that JavaScript and style sheets are enabled and that JavaScript is not blocked by your security preferences.

Note:
If you reset preferences while this Help system is open, click the Reload button on the your browser's navigation toolbar after you reset the preferences.

Man Page Help

You can get man page help for any dc_shell command by choosing Help > Man Pages on the menu bar or entering the following command on the console command line:

```
man command_name
```

To get help on a topic that is a shell command, variable, or variable group, enter

```
help topic_name
```

Design Vision displays the man pages in an HTML-based browser window that lets you view, search, and print man pages for commands, variables, and error messages. You can use the man page viewer to

- Display a man page
- Search for text on the man page you are viewing
- Print the man page you are viewing
- Browse back and forth between man pages you have already viewed

To view a man page in the man page viewer,

1. Choose Help > Man Pages.

   The man page viewer appears. The home page displays a list of links for the different man page categories.

2. Click the category link for the type of man page you want to view: Commands, Variables, or Messages.

   The contents page for the category displays a list of title links for the man pages in that category.

3. Click the title link for the man page you want to view.

For more information about the man page viewer, see the “Viewing Man Pages” topic in online Help.

2

Reading, Constraining, and Optimizing the Design

In this chapter, you will read in the risc_core design VHDL files, apply constraints, and optimize (compile) the design. Allow about one hour to complete the exercises in this chapter.

This chapter includes the following sections:

About the RISC_CORE Design

This tutorial uses a core design of a 16-bit RISC CPU. The top-level design name is RISC_CORE. This is a hierachical design that consists of the following seven instances:

- I_ALU
- I_CONTROL
- I_DATA_PATH
- I_INSTRN_LAT
- I_PRGRM_CNT_TOP
- I_REG_FILE
- I_STACK_TOP

The I_ALU, I_PRGRM_CNT_TOP, and I_STACK_TOP are hierarchical instances, that is, they contain other instances. Each instance (also called cell) is an occurrence of a design loaded in Design Compiler memory.

The instruction size is 32 bits, and the maximum data size is 16 bits. The microprocessor supports 36 different data, arithmetic logic unit (ALU), and control-transfer instructions.

Libraries

You will use the “Synopsys Library Services” 0.25-micron technology library ssc_core. The library units are

- Time unit: 1 nanosecond
- Area unit: 0.5 square microns
- Capacitance unit: 1 picofarad

Directory Structure

As noted in Chapter 1, “Introduction to the Design Compiler Tutorial,” you run this tutorial exercise in the risc_design directory. The subdirectories under the risc_design directory are

- source

  This directory contains the RTL VHDL design files.

- libs

This directory contains the core_typ.db technology library and the core.sdb symbol library.

- scripts

  This directory contains Tcl script files, including the top-level constraints file.

- db

  This directory, which is empty at first, is intended for design files in Synopsys database format (.ddc). You save the unmapped RISC_CORE_GTECH.ddc file and the optimized RISC_CORE_MAPPED.ddc file in this directory.

- work

  This directory, which is empty at first, is used to save design files that you create by using the `analyze` and `elaborate` commands.

Starting the Tutorial Exercise

You start the main tutorial exercise by invoking Design Vision from the risc_design directory.

To start the main exercise,

1. Open a UNIX shell on your terminal.
2. Navigate to the . /dv_tutorial/risc_design directory.
3. At the shell prompt, enter

```
% design_vision
```

The Design Vision window appears. In addition, the design_vision prompt (in dctcl command language) appears in the shell where you started Design Vision and you are ready to begin the tutorial exercise.

Reading the RISC_CORE Designs

To read in the HDL designs and convert them to .ddc format, you can use the `analyze` and `elaborate` commands, or you can use the `read_file` command.

The `analyze` command checks the HDL designs for proper syntax and synthesizable logic, translates the design files into an intermediate format, and stores the intermediate files in the directory you specify (that is, the WORK directory).

The `elaborate` command first checks the intermediate format files before building the design. Then the command determines whether it has the necessary synthetic operators to replace the HDL operators. It also determines correct bus size.

In this tutorial, you use the Analyze and Elaborate menu commands to read in the VHDL designs from the source directory.

Analyzing the RISC_CORE Designs

The VHDL design description includes a VHDL package file and the RTL VHDL design files.

Note:
You must first analyze the VHDL package file, RISCTYPES.vhd, separately; then you analyze the remaining RTL design files. The analyze procedure is the same in both cases.

To analyze the VHDL package,

1. Choose File > Analyze.

   The Analyze Designs dialog box appears.
2. Select VHDL in the Format list.
3. Click the Add button in the dialog box to open the Analyze Designs file browser.
4. Double click the source directory to open it.
5. Select RISCTYPES.vhd from the file name list and click the Select button. The file browser closes.
6. Make sure the WORK library is selected in the Analyze Designs dialog box. (Scroll through the Work library list if necessary.)
7. Click OK.

   The console lists the files that were read into memory, including the libraries standard.sldb, gtech.db, and core_type.db, the Synopsys VHDL primitives, and RISCTYPES.vhd.

After RISCTYPES.vhd is analyzed, you analyze the remaining VHDL design files.

To analyze the VHDL design files,

1. Repeat steps 1 to 4 of the previous procedure.

   Make sure you specify the VHDL format in the Analyze Designs dialog box.
2. In the Analyze Designs file browser, do the following to select all the .vhd files except RISCTYPES.vhd:

- Click the first .vhd file in the file list.
- Shift-click the last (bottom-right) .vhd file in the list so that all .vhd files are selected.
- Control-click RISCTYPES.vhd to remove this file from the list of selected files.

  Note:
  You can use Control-click to remove any extraneous files from a selection list.

- Click the Select button. The file browser closes.

3. Repeat steps 6 and 7 from the previous procedure.

   The intermediate files are now stored in the WORK library, and the console lists the 13 VHDL files that were read into memory.

Equivalent dctcl Commands

```
analyze -format vhdl -lib WORK \
           {./dv_tutorial/risc_design/
source/RISCTYPES.vhd}
analyze -format vhdl -lib WORK \
   {./dv_tutorial/risc_design/source/STACK_TOP.vhd
    ./dv_tutorial/risc_design/source/STACK_MEM.vhd ... }
```

Note:
The second `analyze` command includes the path and file names for the 13 VHDL files.

Elaborating the Top-Level Design

After all the designs are analyzed, you elaborate the top-level design of the hierarchy. All subdesigns are automatically elaborated.

To elaborate the top-level design,

1. Choose File > Elaborate.

   The Elaborate Designs dialog box appears.

2. Select the WORK library.

   The designs in the WORK library appear in the Design list. Click on the arrow beside the Design list to scroll through the list of designs.

3. Select the RISC_CORE(STRUCT) design.
4. Make sure the Re-Analyze Out-of-Date Libraries option is selected.
5. Click OK.

The Elaborate menu command creates generic technology (GTECH) designs from the intermediate format files produced by the Analyze menu command. The console log view lists the subdesigns and inferred devices.

When the elaborate process is done, the current design is the top-level design, RISC_CORE. This design name appears in the instance tree and in the Design list on the toolbar.

The current design is the active design (the design being worked on). Most commands are specific to the current design, that is, they operate within the context of the current design.

Note:

As part of the elaborate process, all designs are *autolinked*. You do not have to manually link the designs.

Equivalent dctcl Command

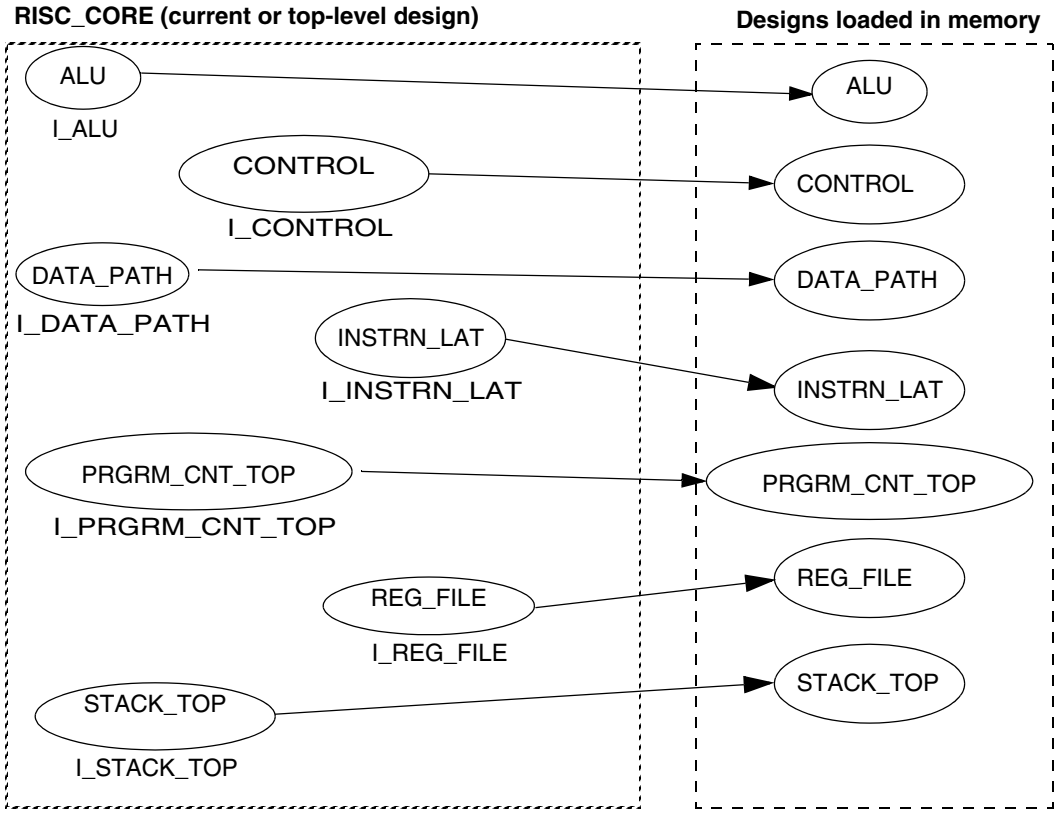
```
elaborate RISC_CORE -arch STRUCT -lib WORK -update
```

Design Hierarchy

The instance tree in the logical hierarchy view lists the RISC_CORE design and its instances (also known as cells). The RISC_CORE design is composed of the following instances: I_ALU, I_CONTROL, I_DATA_PATH, I_INSTRN_LAT, I_PRGRM_CNT_TOP, I_REG_FILE, and I_STACK_TOP. Each instance is an occurrence of a design loaded in Design Compiler memory.

Figure 2-1 shows the relationships among the RISC_CORE design, its instances, and their design references. For example, the I_ALU instance references the ALU design loaded in memory.

Figure 2-1 Instances and Design References



Click the expansion button (plus sign) to expand all instances in the instance tree. Notice that 12 of the instance icons display the letter *G*, which indicates that GTECH designs were built during elaboration.The three hierarchical designs, RISC_CORE, I_PRGRM_CNT_TOP, and I_STACK_TOP, are not elaborated.

Additionally, the STACK_MEM design is referenced by the three instances, I1_STACK_MEM, I2_STACK_MEM, and I3_STACK_MEM. Click I_STACK_TOP (make sure that it's expanded) in the instance tree in the logical hierarchy view. Notice in the objects table in the right pane that STACK_MEM appears as the design reference name for all three instances. (Enlarge the pane as needed to see these design reference names.) Design Compiler automatically resolves these references as part of the compile process.

Applying Design Constraints by Using a Script File

Design constraints define physical conditions that restrict and guide the optimization process. They are applied as attributes with specific values to the designs in memory. If the constraints you define turn out to be unrealistic, you can modify them and recompile the design.

You can set constraints by using the Attribute menu, entering constraint commands directly on the console command line, executing a script file of constraint commands, or using the three methods in any combination.

Important:

For more background information, see Appendix A, "Design Constraints." This appendix includes a copy of the top-level constraints, which you should study before you continue this exercise.

Applying the Top-Level Constraints

In this tutorial, you use the script file method to apply the constraints to the top-level design. The constraints apply to all subdesigns of the design hierarchy.

Note:

If you have run this tutorial before, the designs might still have constraints applied to them. You can use the `reset_design` command or choose Design > Reset Current Design to remove existing constraints. If you are using the `reset_design` command, enter this command on the console command line or edit the constraints script file, top-level.tcl, to include the command at the beginning of the file.

To apply the top-level constraints,

1. Make sure RISC_CORE is the current design.

   The current design name appears in the Design list on the toolbar. You can also determine the current design by entering the `current_design` command on the console command line. The name of the current design is displayed in the console.

2. If RISC_CORE is not the current design, select it from the Design list.

3. Choose File > Execute Script.

   The Execute Script File dialog box appears.

4. Double click the scripts directory to open it.

5. Select top-level.tcl.

   This is the command script file shown in Example A-1 on page A-5.

6. Click Open.

   The constraint commands in the top-level.tcl file are applied to the designs in memory.

   You might see a number of warnings regarding design rule attributes from a driving cell assigned to certain ports. For the purposes of this tutorial, these warnings are not important, and you can ignore them.

Equivalent dctcl Command

```
source ./dv_tutorial/risc_design/scripts/top-level.tcl
```

Checking the Design

You check the constrained, unmapped RISC_CORE design before saving and optimizing it. Use the `check_design` command to check the internal representation of the design and to correct certain design problems. The command issues the appropriate warnings and error messages for design problems it cannot correct.

To check the design for applied constraints,

1. Make sure RISC_CORE is the current design by selecting it in the logical hierarchy view.
2. Choose Design > Check Design.

   The Check Design dialog box appears.
3. Make sure the “Display in detail” option and the “Current level and all sub-designs” option are selected.
4. Click OK.
5. Read the contents of the log view in the console.

   You might see a number of warnings regarding unconnected nets. For the purposes of this tutorial, these warnings are not important, and you can ignore them. Additionally, you will see an informational message that the STACK_MEM design is instantiated three times. Design Compiler automatically resolves these references as part of the compile process.

Equivalent dctcl Command

```
check_design
```

Saving the Unmapped Design With Constraints

If the unmapped, constrained design file passes design checking, you should save the design in .ddc format. By saving the design, you preserve the attribute settings and can restart the design optimization flow from this point. This is useful when you are reoptimizing designs that have a large number of HDL files because you avoid the unnecessary, time-consuming task of re-analyzing and re-elaborating those files.

To save the unmapped RISC_CORE file with attributes in .ddc format,

1. Make sure RISC_CORE is the current design.
2. Choose File > Save As.

   The Save Design As dialog box appears.
3. Double click the db directory to open it.
4. Make sure the “Save all designs in hierarchy” option is selected in the Save Design As dialog box.
5. Enter the file name RISC_CORE_GTECH.ddc in the File name box.
6. Click Save.

   The unmapped file RISC_CORE_GTECH.ddc is saved in the db directory.

Equivalent dctcl Command

```
write -format ddc -hierarchy  -output ./db RISC_CORE_GTECH.ddc
```

Optimizing the Design

After the constraints are applied, you optimize the design by executing the `compile` command. This command has many options (see the `compile` command man page for information); however, in this tutorial, you execute the default compile. The default compile uses the `-map_effort medium` option of the `compile` command.

Note:
You should always begin the design optimization process with a default compilation. For most designs, the default settings produce very good results.

To optimize the unmapped RISC_CORE design,

1. Make sure RISC_CORE is the current design.
2. Choose Design > Compile Design.

   The Compile Design dialog box appears.

3. Make sure of the following settings:
   - The Mapping option “Map design” is selected.

     Make sure that both map effort and area effort are medium.
   - The Design rule option “Fix design rules and optimize mapping” is selected.
   - No other option is selected.
4. Click OK.

The compilation takes about 15 minutes on a 250-MHz machine. To monitor the progress of the compilation, check the messages in the log view of the console.

After optimization finishes, the RISC_CORE design is transferred to the RISC_CORE.ddc database. The mapped design file is a gate-level design implementation that uses components from the technology library.

Equivalent dctcl Command

```
compile -map_effort medium -area_effort medium
```

Saving the Optimized Design

After successfully optimizing the design, you should save the design in .ddc format. By saving and using the mapped design, you can perform timing analysis tasks without having to recompile the design each time.

To save the mapped RISC_CORE file in .ddc format,

1. Make sure RISC_CORE is the current design.

   Check the Design list on the toolbar and change the entry to RISC_CORE if necessary.
2. Choose File > Save As.

   The Save Design As dialog box appears.
3. Open the db directory from the list of directories.
4. Make sure the “Save all designs in hierarchy” option is selected in the Save Design As dialog box.
5. Enter the file name RISC_CORE_MAPPED.ddc in the File name box.
6. Click Save.

   The mapped file RISC_CORE_MAPPED.ddc is saved in the db directory.

Equivalent dctcl Command

```
write -format ddc -hierarchy -output./db/RISC_CORE_MAPPED.ddc
```

Examining the Optimized Design's Structure

With the logical hierarchy view, you can browse the complete hierarchical structure and observe how many hierarchical instances are present, the size of each instance, and whether any DesignWare components have been inferred (used).

To examine the RISC_CORE instance tree,

1. Make sure RISC_CORE is selected in the instance tree of the logical hierarchy view.

   You can use the list above the objects table to view hierarchical cells, all cells, pins and ports, pins of child cells, or nets for the next-lower level of the instantiated cell hierarchy. For each of these, you can view their names and related information.

   If information you want to read is cut off in the display, widen the column as needed. You adjust column widths in the pane by dragging the column split bars.

   The next-lower level of hierarchy includes the following instantiated cells:

   - I_ALU
   - I_CONTROL
   - I_DATA_PATH
   - I_INSTRN_LAT
   - I_PRGRM_CNT_TOP
   - I_REG_FILE
   - I_STACK_TOP

   Note that the three instances I_ALU, I_PRGRM_CNT_TOP, and I_STACK_TOP incorporate lower levels of hierarchy.

2. To view the hierarchical cells of the I_ALU instance, follow these steps:

   - Select the I_ALU instance in the left pane.
   - Select the Cells (Hierarchical) option in the list above the objects table.

     The objects table changes to show information about the instantiated hierarchical cells that constitute the I_ALU cell. Notice that these cells reference DesignWare carry-lookahead-adder components, as indicated by the ALU_DW01_ part of the reference names.

- Select any component of the I_ALU instance in the left pane.

  Because these DesignWare components contain no lower level of hierarchy, no information appears in the objects table. However, if you select the Cells (All) option in the list above the objects table for any component, you can view the leaf cells that constitute the component.

You can repeat the preceding steps for any instance in the instance tree.The objects table will show the appropriate information for the cells at the next-lower level of the hierarchy.

Quitting the Tutorial After Optimization

With the compiled RISC_CORE.ddc design saved in the design database, you can quit the tutorial and restart it from this point at a later time, or you can continue by going on to the timing analysis of Chapter 3, “Analyzing Timing and Area Results.”

To quit the session, choose File > Exit and click OK in the warning box.

Equivalent dctcl Command

```
exit
```

3

Analyzing Timing and Area Results

After the design is optimized, you analyze the timing and area results to determine if the design meets the constraints and optimization goals or if modifications are necessary. Design Vision provides a number of visualization tools for performing timing analysis, including endpoint slack histogram, path slack histograms, and annotated path schematics. In addition, you can generate constraint analysis reports on timing and area results, as well as other reports.

Note:

Slack is defined as the time difference between the timing goal for a path and its actual timing. Paths that meet the design's timing goals have positive slack values; those that do not have negative slack values.

Allow about one hour to complete the exercises in this chapter.

This chapter includes the following sections:

Restarting the Tutorial With the Optimized Design

If you quit the tutorial at the end of Chapter 2, restart the tutorial from the risc_design directory, using the optimized design RISC_CORE_MAPPED.ddc.

To restart the tutorial with the mapped RISC_CORE.ddc design,

1. Open a UNIX shell on your terminal.
2. Navigate to the risc_design directory.
3. At the shell prompt, enter

   ```
   % design_vision
   ```

   The Design Vision window opens.
4. Choose File > Read.

   The Read Designs dialog box appears.
5. Double click the db directory to open it.
6. Select RISC_CORE_MAPPED.ddc.
7. Click Open.

   The compiled RISC_CORE top-level design and all its compiled subdesigns are read into memory. You can confirm that the designs have been loaded by checking the instance tree in the logical hierarchy view.

Equivalent dctcl Command

```
read_file -format ddc {./db/RISC_CORE_MAPPED.ddc}
```

Generating the Critical Path Timing Report

You can begin your analysis of the optimization results by examining timing reports for the top-level design. The timing report provides detailed timing information for the paths of the design. You can usually determine the cause of timing violations by investigating this data.

You generate a timing report by using the Report Timing command of the Timing menu.

To generate the critical path timing report for the RISC_CORE design,

1. Select RISC_CORE in the logical hierarchy view.
2. Choose Timing > Report Timing Path.

   The Report Timing Paths dialog box appears. No path information is displayed in the Paths boxes. Do not change the default settings, which are currently set as follows:

   - Worst path per endpoint, 1
   - Max paths per group, 1
   - Path type, full
   - Delay type, max

   The results are sorted by group. With these settings, the timing report provides information only for the design's critical path (the path with the worst slack value).
3. Click OK.

   The worst slack timing report for the RISC_CORE design appears in a new report view and the console log view. Maximize the report view or the console log view to scroll through the report.

   The worst slack timing report begins by showing the default options, operating conditions, technology library, wire load model mode, and startpoint and endpoint of the worst timing path in the design. The path group and type are displayed. This information is followed by a table of incremental timing contributions from each of the cells (approximately 30, almost all through the ALU instance) in the path and the cumulative timing or data arrival time, which is approximately 3.73 ns. (Your value might be slightly different.)

   The worst slack timing report shows that slack has been met.

The timing report is similar to the one shown in Example 3-1.

Example 3-1 Worst Slack Timing Report

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
        -sort_by group
Design : RISC_CORE
Version: V-2003.12
Date   : Mon Oct 20 12:58:06 2003
****************************************

Operating Conditions: typ_0_1.98   Library: ssc_core
Wire Load Model Mode: enclosed
  Startpoint: I_DATA_PATH/Oprnd_B_reg[0]
              (rising edge-triggered flip-flop clocked by my_clock)
  Endpoint: I_ALU/Zro_Flag_reg
            (rising edge-triggered flip-flop clocked by my_clock)
  Path Group: my_clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  RISC_CORE          10KGATES              ssc_core
  ALU                5KGATES               ssc_core
  ALU_DW01_inc_16_2  5KGATES
ssc_core
  ALU_DW01_add_16_0  5KGATES               ssc_core

  Point                                                        Incr       Path
  ------------------------------------------------------------------------------
  I_DATA_PATH/Oprnd_B_reg[0]/CLK (fdef1a9)                     0.00       0.00 r
  I_DATA_PATH/Oprnd_B_reg[0]/Q (fdef1a9)                       0.42       0.42 r
  I_DATA_PATH/Oprnd_B[0] (DATA_PATH)                           0.00       0.42 r
  I_ALU/Oprnd_B[0] (ALU)                                       0.00       0.42 r
  I_ALU/U206/Y (buf1a27)                                       0.16       0.58 r
  I_ALU/add_75/plus/A[0] (ALU_DW01_inc_16_2)                   0.00       0.58 r
  I_ALU/add_75/plus/U18/Y (inv1a9)                             0.07       0.66 f
  I_ALU/add_75/plus/U42/Y (or4a6)                              0.23       0.89 f
  I_ALU/add_75/plus/U24/Y (inv1a15)                            0.08       0.97 r
  I_ALU/add_75/plus/U23/Y (and2a9)                             0.21       1.19 r
  I_ALU/add_75/plus/U10/Y (and2a3)                             0.19       1.38 r
  I_ALU/add_75/plus/U27/Y (xor2a6)                             0.22       1.60 f
  I_ALU/add_75/plus/SUM[9] (ALU_DW01_inc_16_2)                 0.00       1.60 f
  I_ALU/add_75/plus_172/B[9] (ALU_DW01_add_16_0)               0.00       1.60 f
  I_ALU/add_75/plus_172/U144/Y (or2a15)                        0.19       1.79 f
  I_ALU/add_75/plus_172/U33/Y (inv1a6)                         0.09       1.88 r
  I_ALU/add_75/plus_172/U128/Y (or2a3)                         0.21       2.09 r
  I_ALU/add_75/plus_172/U124/Y (or2a3)                         0.21       2.29 r
  I_ALU/add_75/plus_172/U148/Y (ao1f15)                        0.28       2.57 f
  I_ALU/add_75/plus_172/U122/Y (xor2a6)                        0.18       2.76 f
  I_ALU/add_75/plus_172/SUM[11] (ALU_DW01_add_16_0)            0.00       2.76 f
  I_ALU/U536/Y (and2a15)                                       0.12       2.88 f
  I_ALU/U535/Y (or4a6)                                         0.21       3.09 f
  I_ALU/U472/Y (pclk1b6)                                       0.07       3.16 r
  I_ALU/U199/Y (and3a15)                                       0.17       3.34 r
  I_ALU/U202/Y (inv1a9)                                        0.06       3.40 f
  I_ALU/U499/Y (oa1f6)                                         0.11       3.51 r
```

```
I_ALU/U492/Y (and4a6)                                    0.22       3.73 r
I_ALU/Zro_Flag_reg/D (fdef1a9)                           0.00       3.73 r
data arrival time                                                   3.73
clock my_clock (rise edge)                               4.00       4.00
clock network delay (ideal)                              0.00       4.00
clock uncertainty                                       -0.14       3.86
I_ALU/Zro_Flag_reg/CLK (fdef1a9)                         0.00       3.86 r
library setup time                                      -0.13       3.73
data required time                                                  3.73
--------------------------------------------------------------------------
data required time                                                  3.73
data arrival time                                                  -3.73
--------------------------------------------------------------------------
slack (MET)                                                         0.00
```

Equivalent dctcl Command

```
report_timing -path full -delay max -nworst 1 \
   -max_paths
1 -significant_digits 2 -sort_by group
```

Analyzing Endpoint Slack

You can obtain timing slack information for all path endpoints in the design by using endpoint slack histograms. Endpoint slack histograms show a distribution of timing slack values for all endpoints in a design. The slack distribution provides an overall picture of how close the design is to meeting timing requirements.

The endpoint slack histogram is split into two panes: the histogram graphic on the left and an endpoints-slack table on the right.

Displaying the Endpoint Slack Histogram

To display the endpoint slack histogram for the RISC_CORE design,

1. Select RISC_CORE in the logical hierarchy view.
2. Choose Timing > Endpoint Slack.

   The Endpoint Slack dialog box appears.
3. Make sure “Delay type” is max and “Number of bins” is 8.
4. Click OK or Apply.

   The endpoint slack histogram is displayed. The eight bins represent the number of endpoints (y-axis) versus their slack values (x-axis). The number at the top of the tallest bin indicates the number of endpoints in the bin. Green bins (on the positive side of 0) contain endpoints in the design that passed their constraints. Red bins (on the negative side of 0) contain endpoints in the design that failed their constraints.

If you hold the pointer over a bin, an InfoTip displays the number of endpoints and the range of slack values in the bin.

Displaying the Endpoints-Slack Table

You can display detailed endpoint slack information for the paths of a bin in the table to the right of the endpoint slack histogram.

To display detailed endpoint slack information,

1. Click the highest bar of the histogram.

   The bin color changes to yellow to indicate that it has been selected, and all the path endpoints belonging to the bin together with their slack values are displayed in the endpoints-slack table.
2. Click the histogram bar for the bin with the smallest slack values.

   The bin changes to yellow, and the path endpoints for this bin appear in the table. (The previously selected bin color changes back to its original color.)
3. Use the same steps to investigate other endpoint slack bins.

Generating a Timing Report From the Endpoint Slack Histogram for a Specific Endpoint

You can obtain timing reports for any endpoint of the histogram by specifying the endpoint in the Report Timing Paths dialog box. However, an easier way to specify the endpoint is to select it in the endpoints-slack table.

To generate a timing report for a specific endpoint,

1. Select the slack bin with the worst slack values from the endpoint slack histogram.
2. Select one of the endpoints in the endpoints-slack table.
3. Choose Timing > Report Timing.

   The Report Timing Paths dialog box is displayed.
4. Click the Selection[3] button beside the To box in the Paths area. (If the endpoint that you selected is a port, the Selection[3] button is dimmed. Change from pin to port in the To list and then click the Selection[3] button.)

   The endpoint that you selected in the endpoints-slack table appears in the To box.

5. Click OK.

   A timing report for the worst path of the specified endpoint appears in a new report view and the console log view.

Equivalent dctcl Command

```
report_timing -to
      {I_ALU/Zro_Flag_reg/D} -path full -delay
      max -nworst 1 -max_paths 1
      -significant_digits 2 -sort_by group
```

Analyzing Area Results

After meeting the design timing goals, you should check whether the area constraint was met. The maximum area constraint was set to 380,000 area units in the constraints script file. (Recall that the technology library area units are 0.5 square microns.) You can generate an area report to find out the total area and total cell area, as well as the combinational, noncombinational, and net interconnect areas.

To generate an area report for the RISC_CORE design,

1. Make sure RISC_CORE is the current design, or select the RISC_CORE object in the instance tree of the logical hierarchy view.
2. Choose Design > Report Area.
3. Click OK or Apply.

   The area report is displayed in a new report view and the console log view. The total area of the design is approximately 350,000 area units, which is less than the constraint value of 380,000 area units. The optimized design meets the area goal.

   The report also provides the individual area contributions to the total area. The total cell area is the sum of the combinational and noncombinational areas, and the total area is the sum of the net interconnect area and the total cell area. In this example, the net interconnect area is about two-thirds of the total area.

Equivalent dctcl Command

```
report_area -nosplit
```

Quitting the Tutorial After Timing and Area Analysis

After timing and area analysis, you can quit the tutorial and restart it from this point at a later time, or you can continue by going on to Chapter 4, “Modifying Constraints and Reoptimizing the Design.”

To quit the session, choose File > Exit and click OK in the warning box.

Equivalent dctcl Command

```
exit
```

4

Modifying Constraints and Reoptimizing the Design

In this chapter you will read in the mapped RISC_CORE design, tighten constraints, modify a subdesign structure, analyze timing results, and recompile the design by using the incremental compile option.

This chapter includes the following sections:

Restarting the Tutorial With the Optimized Design

If you quit the tutorial at the end of Chapter 3, restart the tutorial from the risc_design directory, using the optimized design RISC_CORE_MAPPED.ddc.

To restart the tutorial with the mapped RISC_CORE.ddc design,

1. Open a UNIX shell on your terminal.
2. Navigate to the risc_design directory.
3. At the shell prompt, enter

   ```
   % design_vision
   ```

   The Design Vision window opens.
4. Choose File > Read.

   The Read Designs dialog box appears.
5. Double click the db directory to open it.
6. Select RISC_CORE_MAPPED.ddc.
7. Click Open.

   The compiled RISC_CORE top-level design and all its compiled subdesigns are read into memory. You can confirm that the designs have been loaded by checking the logical hierarchy view.

Equivalent dctcl Command

```
read_file -format ddc \
      {./db/RISC_CORE_MAPPED.ddc}
```

Modifying Constraints

Constraints are the timing and environmental restrictions under which Design Compiler performs synthesis. For more information on constraints, see “Design Constraints” in Appendix A.

In this exercise, you modify the output delay on the RESULT_DATA bus.

Output delay is a constraint that specifies the minimum or maximum amount of delay from an output port to the sequential element that captures data from the output port. It establishes the time at which data must be available at the output port of a launching element to meet the setup and hold requirements of the capturing element. Output delays of a design are defined relative to the clocks of the design.

To change the output delay on the RESULT_DATA bus from 0.5 to 1.1 ns, follow these steps:

1. Select RISC_CORE in the logical hierarchy view.
2. To check the current output delay on the RESULT_DATA bus, follow these steps:
   - Select Pins/Ports in the list above the objects table.

     All the pins and ports of the RISC_CORE design are listed in the objects table.
   - Select the RESULT_DATA bus by Shift-clicking the 16 RESULT_DATA ports.
   - Choose Design > Report Ports.

     The Report Ports dialog box appears.
   - Click the Selection button beside the Ports field.

     The RESULT_DATA ports appear in the Ports box.
   - Select the Verbose Report option.
   - Click OK or Apply.

     The Ports report is displayed in a new report view and the console log view. Maximize this report.

     Scroll through the report until you locate the section on output delay. The RESULT_DATA bus has an output delay of 0.5 ns.
3. To change output delay on the RESULT_DATA bus, follow these steps:
   - Make sure the RESULT_DATA bus is selected in the right pane of the logical hierarchy view.
   - Choose Attributes > Operating Environment > Output Delay.
   - In the Output Delay dialog box, select my_clock in the Relative to clock list.
   - Make sure the “Same rise and fall” option is checked, and select the “Add delay” option.
   - Enter 1.1 in the Max rise box.
   - Click OK or Apply.

     A delay of 1.1 ns is now applied to the RESULT_DATA bus. You can check that this delay has been applied by following step 2 of these instructions.

Equivalent dctcl Commands

```
report_port -verbose [get_ports {RESULT_DATA*}]

set_output_delay 1.1 -clock my_clock -add_delay -max \
  -rise {RESULT_DATA[*]}

set_output_delay 1.1 -clock my_clock -add_delay -max \
  -fall {RESULT_DATA[*]}
```

Modifying a Subdesign Structure

Because Design Compiler does not optimize across hierarchical boundaries, you might want to remove the hierarchy within certain designs. By doing so, you might be able to improve timing results.

Removing a level of hierarchy is known as ungrouping. Ungrouping merges subdesigns of a given level of the hierarchy into the parent cell or design.

The critical path timing report that you generated in Chapter 3 for the RISC_CORE design shows that the paths with the worst slack values involve instances in the I_ALU instance. An instance is an occurrence of a design loaded in memory; it is also known as a cell. The I_ALU instance possesses hierarchical structure, containing from 6 to 10 child instances.

You remove this hierarchy by ungrouping the I_ALU instance. The cells of the I_ALU instance are then optimized individually instead of as a group.

Ungrouping the Cells Under the I_ALU Instance Hierarchy

Using the logical hierarchy view, you can see that the inferred designs for the cells in the I_ALU instance are all DesignWare components with reference names beginning ALU_DW01_.

To ungroup the I_ALU instance,

1. Expand the I_ALU instance in the instance tree of the logical hierarchy view by clicking the expansion button (plus sign).
2. Select all child instances of the I_ALU instance by clicking and Shift-clicking these objects in the instance tree.
3. Choose Hierarchy > Ungroup.

   The Ungroup dialog box appears.
4. Make sure that the “Selected cells” option is selected, and select the “Ungroup all levels” option in the Ungroup dialog box.

5. Click OK.

   The I_ALU instance is now flat. You do not see cell names, reference names, cell paths, and cell area information if you click the I_ALU instance. However, if you select I_ALU in the instance tree, generate its schematic (by choosing Schematic > New Design Schematic View), and then magnify the schematic as needed, you can see a large number of leaf cells that constitute the I_ALU instance.

Equivalent dctcl Commands

```
current_instance /RISC_CORE/I_ALU
ungroup -all -flatten
```

Analyzing Timing Results

Using the timing analysis methods described in Chapter 3, “Analyzing Timing and Area Results,” you will investigate the effect of changing the output delay on the RESULT_DATA bus.

Note:

The following analysis quotes specific results for a particular set of worst timing paths.Your worst timing paths and their slack values might differ, but there will be no essential difference between your results and these. You will be able to apply the same procedure to your paths and achieve similar results.

Generating the Critical Path Timing Report

To generate the critical path timing report for the RISC_CORE design,

1. Select RISC_CORE in the logical hierarchy view.
2. Choose Timing > Report Timing.

   The Report Timing Paths dialog box appears. No path information is displayed in the Paths boxes. Do not change the default settings, which are currently set as follows:

   - Worst path per endpoint, 1
   - Max paths per group, 1
   - Path type, full
   - Delay type, max

   The results are sorted by group. With these settings, the timing report provides information only for the design’s critical path (the path with the worst slack value).

3. Click OK.

   The worst slack timing report for the RISC_CORE design is displayed in a new report view and the console log view. The report shows a negative slack of -0.31 ns.

Analyzing the Critical Path Timing Report

The worst slack timing report shows a table of incremental timing contributions from each of the cells in the worst timing path and the cumulative timing or data arrival time, which is approximately 3.07 ns. (Your value might be slightly different.)

The data arrival time is computed relative to the rising edge of the my_clock clock pulse at time 0, which is when the data is launched from the I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/CLK pin. (Your startpoint might be different.)

The data is captured at the RESULT_DATA[0] (out) pin on the rising edge of the next clock pulse. (Your endpoint might be different.) If no corrections were necessary, the data would be required to arrive within the clock period. But both the clock uncertainty of 0.14 ns and the output external delay of 1.10 ns must be taken into account. Therefore, the data has to arrive at the RESULT_DATA[0] (out) pin 1.24 ns before the rising edge clock pulse.

Comparing the required data arrival time with the actual data arrival time leads to a slack value of approximately -0.31 ns. The data arrives late, and therefore this path does not meet the timing goals.

By examining the slack contributions of the individual cells along the timing path, you can often locate the problem cells.

Generating a Timing Report for Multiple Paths

You can obtain timing information for more than one path. For example, repeat steps 2 and 3 in the preceding procedure, but change the settings for the following:

- Max paths per group, 5

The report now includes timing information for the five worst paths in the design.

Equivalent dctcl Command

```
report_timing -path full -delay max -nworst 1 \
   -max_paths 5 -significant_digits 2 -sort_by group
```

The timing report is similar to the one shown in Example 4-1.

Example 4-1 Timing Report for Multiple Paths

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 5
        -sort_by group
Design : RISC_CORE
Version: V-2003.12
Date   : Mon Oct 27 15:09:40 2003
****************************************

Operating Conditions: typ_0_1.98   Library: ssc_core
Wire Load Model Mode: enclosed
  Startpoint: I_INSTRN_LAT/Crnt_Instrn_2_reg[24]
              (rising edge-triggered flip-flop clocked by my_clock)
  Endpoint: RESULT_DATA[0]
            (output port clocked by my_clock)
  Path Group: my_clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  RISC_CORE          10KGATES              ssc_core
  DATA_PATH          5KGATES               ssc_core
  REG_FILE           5KGATES               ssc_core

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock my_clock (rise edge)                              0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/CLK (fdef1a9)        0.00       0.00 r
  I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/Q (fdef1a9)          0.43       0.43 r
  I_INSTRN_LAT/Crnt_Instrn_2[24] (INSTRN_LAT)             0.00       0.43 r
  I_DATA_PATH/Crnt_Instrn[24] (DATA_PATH)                 0.00       0.43 r
  I_DATA_PATH/U176/Y (inv1a9)                             0.08       0.51 f
  I_DATA_PATH/U80/Y (or3a2)                               0.36       0.88 f
  I_DATA_PATH/U175/Y (inv1a9)                             0.16       1.03 r
  I_DATA_PATH/U78/Y (mx2a3)                               0.41       1.45 r
  I_DATA_PATH/Addr_A[1] (DATA_PATH)                       0.00       1.45 r
  I_REG_FILE/Addr_A[1] (REG_FILE)                         0.00       1.45 r
  I_REG_FILE/U116/Y (inv1a3)                              0.17       1.62 r
  I_REG_FILE/U348/Y (or2a3)                               0.74       2.36 r
  I_REG_FILE/U349/Y (or2a3)                               0.24       2.59 r

  I_REG_FILE/U120/Y (and4a3)                              0.28       2.87 r
  I_REG_FILE/U135/Y (inv1a3)                              0.20       3.07 f
  I_REG_FILE/RegPort_A[0] (REG_FILE)                      0.00       3.07 f
  RESULT_DATA[0] (out)                                    0.00       3.07 f
  data arrival time                                                  3.07

  clock my_clock (rise edge)                              4.00       4.00
  clock network delay (ideal)                             0.00       4.00
  clock uncertainty                                      -0.14       3.86
  output external delay                                  -1.10       2.76
  data required time                                                 2.76
  --------------------------------------------------------------------------
  data required time                                                 2.76
```

```
data arrival time                                                      -3.07
---------------------------------------------------------------------------
slack (VIOLATED)                                                       -0.31

Startpoint: I_INSTRN_LAT/Crnt_Instrn_2_reg[24]
            (rising edge-triggered flip-flop clocked by my_clock)
Endpoint: RESULT_DATA[1]
          (output port clocked by my_clock)
Path Group: my_clock
Path Type: max

Des/Clust/Port     Wire Load Model       Library
------------------------------------------
RISC_CORE          10KGATES              ssc_core
DATA_PATH          5KGATES               ssc_core
REG_FILE           5KGATES               ssc_core

Point                                                    Incr       Path
---------------------------------------------------------------------------
clock my_clock (rise edge)                               0.00       0.00
clock network delay (ideal)                              0.00       0.00
I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/CLK (fdef1a9)         0.00       0.00 r
I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/Q (fdef1a9)           0.43       0.43 r
I_INSTRN_LAT/Crnt_Instrn_2[24] (INSTRN_LAT)              0.00       0.43 r
I_DATA_PATH/Crnt_Instrn[24] (DATA_PATH)                  0.00       0.43 r
I_DATA_PATH/U176/Y (inv1a9)                              0.08       0.51 f
I_DATA_PATH/U80/Y (or3a2)                                0.36       0.88 f
I_DATA_PATH/U175/Y (inv1a9)                              0.16       1.03 r
I_DATA_PATH/U78/Y (mx2a3)                                0.41       1.45 r
I_DATA_PATH/Addr_A[1] (DATA_PATH)                        0.00       1.45 r
I_REG_FILE/Addr_A[1] (REG_FILE)                          0.00       1.45 r
I_REG_FILE/U116/Y (inv1a3)                               0.17       1.62 r
I_REG_FILE/U348/Y (or2a3)                                0.74       2.36 r
I_REG_FILE/U354/Y (or2a3)                                0.24       2.59 r
I_REG_FILE/U117/Y (and4a3)                               0.28       2.87 f

I_REG_FILE/U274/Y (inv1a3)                               0.20       3.07 f
I_REG_FILE/RegPort_A[2] (REG_FILE)                       0.00       3.07 f
RESULT_DATA[2] (out)                                     0.00       3.07 f
data arrival time                                                   3.07

clock my_clock (rise edge)                               4.00       4.00
clock network delay (ideal)                              0.00       4.00
clock uncertainty                                       -0.14       3.86
output external delay                                   -1.10       2.76
data required time                                                  2.76
---------------------------------------------------------------------------
data required time                                                  2.76
data arrival time                                                  -3.07
---------------------------------------------------------------------------
slack (VIOLATED)                                                   -0.31

Startpoint: I_INSTRN_LAT/Crnt_Instrn_2_reg[24]
            (rising edge-triggered flip-flop clocked by my_clock)
Endpoint: RESULT_DATA[2]
          (output port clocked by my_clock)
Path Group: my_clock
Path Type: max
```

```
Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
RISC_CORE          10KGATES              ssc_core
DATA_PATH          5KGATES               ssc_core
REG_FILE           5KGATES               ssc_core

Point                                                   Incr       Path
-------------------------------------------------------------------------
clock my_clock (rise edge)                              0.00       0.00
clock network delay (ideal)                             0.00       0.00
I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/CLK (fdef1a9)        0.00       0.00 r
I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/Q (fdef1a9)          0.43       0.43 r
I_INSTRN_LAT/Crnt_Instrn_2[24] (INSTRN_LAT)             0.00       0.43 r
I_DATA_PATH/Crnt_Instrn[24] (DATA_PATH)                 0.00       0.43 r
I_DATA_PATH/U176/Y (inv1a9)                             0.08       0.51 f
I_DATA_PATH/U80/Y (or3a2)                               0.36       0.88 f
I_DATA_PATH/U175/Y (inv1a9)                             0.16       1.03 r
I_DATA_PATH/U78/Y (mx2a3)                               0.41       1.45 f
I_DATA_PATH/Addr_A[1] (DATA_PATH)                       0.00       1.45 f
I_REG_FILE/Addr_A[1] (REG_FILE)                         0.00       1.45 f
I_REG_FILE/U116/Y (inv1a3)                              0.17       1.62 r
I_REG_FILE/U348/Y (or2a3)                               0.74       2.36 r
I_REG_FILE/U358/Y (or2a3)                               0.24       2.59 r
I_REG_FILE/U131/Y (and4a3)                              0.28       2.87 r
I_REG_FILE/U140/Y (inv1a3)                              0.20       3.07 f
I_REG_FILE/RegPort_A[2] (REG_FILE)                      0.00       3.07 f
RESULT_DATA[2] (out)                                    0.00       3.07 f
data arrival time                                                  3.07

clock my_clock (rise edge)                              4.00       4.00
clock network delay (ideal)                             0.00       4.00
clock uncertainty                                      -0.14       3.86
output external delay                                  -1.10       2.76
data required time                                                 2.76
-------------------------------------------------------------------------
data required time                                                 2.76
data arrival time                                                 -3.07
-------------------------------------------------------------------------
slack (VIOLATED)                                                  -0.31

Startpoint: I_INSTRN_LAT/Crnt_Instrn_2_reg[24]
            (rising edge-triggered flip-flop clocked by my_clock)
Endpoint: RESULT_DATA[3]
          (output port clocked by my_clock)
Path Group: my_clock
Path Type: max

Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
RISC_CORE          10KGATES              ssc_core
DATA_PATH          5KGATES               ssc_core
REG_FILE           5KGATES               ssc_core

Point                                                   Incr       Path
-------------------------------------------------------------------------
clock my_clock (rise edge)                              0.00       0.00
clock network delay (ideal)                             0.00       0.00
```

```
I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/CLK (fdef1a9)        0.00       0.00 r
I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/Q (fdef1a9)          0.43       0.43 r
I_INSTRN_LAT/Crnt_Instrn_2[24] (INSTRN_LAT)             0.00       0.43 r
I_DATA_PATH/Crnt_Instrn[24] (DATA_PATH)                 0.00       0.43 r
I_DATA_PATH/U176/Y (inv1a9)                             0.08       0.51 f
I_DATA_PATH/U80/Y (or3a2)                               0.36       0.88 f
I_DATA_PATH/U175/Y (inv1a9)                             0.16       1.03 r
I_DATA_PATH/U78/Y (mx2a3)                               0.41       1.45 f
I_DATA_PATH/Addr_A[1] (DATA_PATH)                       0.00       1.45 f
I_REG_FILE/Addr_A[1] (REG_FILE)                         0.00       1.45 f
I_REG_FILE/U116/Y (inv1a3)                              0.17       1.62 r
I_REG_FILE/U348/Y (or2a3)                               0.74       2.36 r
I_REG_FILE/U362/Y (or2a3)                               0.24       2.59 r
I_REG_FILE/U125/Y (and4a3)                              0.28       2.87 r
I_REG_FILE/U141/Y (inv1a3)                              0.20       3.07 f
I_REG_FILE/RegPort_A[3] (REG_FILE)                      0.00       3.07 f
RESULT_DATA[3] (out)                                    0.00       3.07 f
data arrival time                                                  3.07

clock my_clock (rise edge)                              4.00       4.00
clock network delay (ideal)                             0.00       4.00
clock uncertainty                                      -0.14       3.86
output external delay                                  -1.10       2.76
data required time                                                 2.76
--------------------------------------------------------------------------
data required time                                                 2.76
data arrival time                                                 -3.07
--------------------------------------------------------------------------
slack (VIOLATED)                                                  -0.31

Startpoint: I_INSTRN_LAT/Crnt_Instrn_2_reg[24]
            (rising edge-triggered flip-flop clocked by my_clock)
Endpoint: RESULT_DATA[4]
          (output port clocked by my_clock)
Path Group: my_clock
Path Type: max

Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
RISC_CORE          10KGATES              ssc_core
DATA_PATH          5KGATES               ssc_core
REG_FILE           5KGATES               ssc_core

Point                                                   Incr       Path
--------------------------------------------------------------------------
clock my_clock (rise edge)                              0.00       0.00
clock network delay (ideal)                             0.00       0.00
I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/CLK (fdef1a9)        0.00       0.00 r
I_INSTRN_LAT/Crnt_Instrn_2_reg[24]/Q (fdef1a9)          0.43       0.43 r
I_INSTRN_LAT/Crnt_Instrn_2[24] (INSTRN_LAT)             0.00       0.43 r
I_DATA_PATH/Crnt_Instrn[24] (DATA_PATH)                 0.00       0.43 r
I_DATA_PATH/U176/Y (inv1a9)                             0.08       0.51 f
I_DATA_PATH/U80/Y (or3a2)                               0.36       0.88 f
I_DATA_PATH/U175/Y (inv1a9)                             0.16       1.03 r
I_DATA_PATH/U78/Y (mx2a3)                               0.41       1.45 f
I_DATA_PATH/Addr_A[1] (DATA_PATH)                       0.00       1.45 f
I_REG_FILE/Addr_A[1] (REG_FILE)                         0.00       1.45 f
I_REG_FILE/U116/Y (inv1a3)                              0.17       1.62 r
```

```
I_REG_FILE/U348/Y (or2a3)                              0.74       2.36 r
I_REG_FILE/U366/Y (or2a3)                              0.24       2.59 r
I_REG_FILE/U124/Y (and4a3)                             0.28       2.87 r
I_REG_FILE/U136/Y (inv1a3)                             0.20       3.07 f
I_REG_FILE/RegPort_A[4] (REG_FILE)                     0.00       3.07 f
RESULT_DATA[4] (out)                                   0.00       3.07 f
data arrival time                                                 3.07

clock my_clock (rise edge)                             4.00       4.00
clock network delay (ideal)                            0.00       4.00
clock uncertainty                                     -0.14       3.86
output external delay                                 -1.10       2.76
data required time                                                2.76
--------------------------------------------------------------------------
data required time                                                2.76
data arrival time                                                -3.07
--------------------------------------------------------------------------
slack (VIOLATED)                                                 -0.31
```

Analyzing Endpoint Slack

In this exercise, you display the endpoint slack histogram for the RISC_CORE design and generate a timing report for a specific endpoint. Endpoint slack histograms provide a high-level overview of the timing quality of your design. You create an endpoint slack histogram to identify endpoints that failed their constraints.

Displaying the Endpoint Slack Histogram

To display the endpoint slack histogram for the RISC_CORE design,

1. Select RISC_CORE in the logical hierarchy view.
2. Choose Timing > Endpoint Slack.

   The Endpoint Slack dialog box appears.
3. Make sure “Delay Type” is set to max and the “Number of bins” is set to 8.
4. Click OK or Apply.
5. Place the pointer over the worst slack bin (red) and click.

   The bin color changes to yellow to indicate that it has been selected, and all the path endpoints belonging to the bin together with their slack values are displayed in the endpoints-slack table.

   The endpoints-slack table to the right of the histogram shows that the endpoint pin names with the worst negative slack values belong to the RESULT_DATA bus.

Generating a Timing Report for a Specific Endpoint

You can generate a timing report for any of the endpoints of the RESULT_DATA bus, which has 16 endpoints. To do so, follow these steps:

1. Select one of the RESULT_DATA endpoints in the endpoints-slack table.
2. Choose Timing > Report Timing.

   The Report Timing Paths dialog box is displayed.
3. Select the port object from the To list in the Paths area.
4. Click the Selection[3] button beside the To box.

   The endpoint that you selected in the endpoints-slack table appears in the To box.
5. Click OK.

   A timing report for the worst path of the specified RESULT_DATA endpoint appears in a new report view and the console log view. The RESULT_DATA endpoint has a negative slack of about -0.31 ns.

Equivalent dctcl Command

```
report_timing -to {RESULT_DATA[2]} -path full -delay max \
      -nworst 1 -max_paths 5 -significant_digits 2 -sort_by group
```

Reoptimizing the RISC_CORE Design and Comparing Timing Results

After you have changed the output delay on the RESULT_DATA bus and ungrouped the I_ALU instance, you can use the incremental compile function to reoptimize the top-level design. An incremental compile affects only instances in which you have made changes, enabling you to incrementally improve your design. It is often successful in reducing the worst negative slack to zero.

To reoptimize the RISC_CORE design,

1. Select RISC_CORE in the logical hierarchy view.
2. Choose Design > Compile Design.

   The Compile Design dialog box appears.
3. Make sure of the following settings:
   - The Mapping option “Map design” is selected.

     Make sure that both map effort and area effort are high.

- The Design rule option “Fix design rules and optimize mapping” is selected.
- The Compile option “Incremental mapping” is selected.
- None of the other options are selected.

4. Click OK.
5. When the incremental compilation finishes, repeat the timing analysis on the recompiled RISC_CORE design by generating an endpoint slack histogram and a timing report.

   The histogram and the timing report show that no paths exhibit negative slack and that the timing results are generally improved.

Equivalent dctcl Commands

```
current_design ./dv_tutorial/risc_design/RISC_CORE.ddc:RISC_CORE
compile -map_effort high -area_effort high -incremental_mapping
```

Quitting the Tutorial

To save the recompiled RISC_CORE in .ddc format,

1. Make sure RISC_CORE is the current design.

   Check the Design list on the toolbar, and change the selection to RISC_CORE if necessary.

2. Choose File > Save As.

   The Save Design As dialog box appears.

3. Double click the db directory to open it.
4. Make sure the “Save all designs in hierarchy” option is selected in the Save Design As dialog box.
5. Enter the file name RISC_CORE_MAPPED.ddc in the File Name box.
6. Click Save.

   The mapped file RISC_CORE_MAPPED.ddc is saved in the db directory.

You must quit this tutorial exercise before going on to the exercises in Chapter 5. In those exercises you will invoke Design Vision and Design Compiler from different directories.

Equivalent dctcl Commands

```
write -format ddc -hierarchy -output \
      ./dv_tutorial/risc_design/db/RISC_CORE_MAPPED.ddc
exit
```

5

Working With False Paths and Multiple Clocks

In this chapter you will do two exercises. In the first exercise, you learn how to use false paths to improve timing results in your design. You compile the design, specify certain paths as false paths, reoptimize the design, and analyze the results. In the second exercise, you investigate the effect of multiple clocks on your design. You define multiple clocks for the design, optimize the design, and analyze the timing results. Allow about one hour to complete the exercises in this chapter.

This chapter includes the following sections:

Specifying False Paths to Improve Timing

In some designs there are paths that need not, or should not, be taken into account by the Design Compiler optimization processes. Such paths are not timing critical, can mask other paths that must be considered during optimization, or never occur in normal operation. An example of such a path is one between two multiplexed blocks that are never enabled at the same time.

You can exclude such paths from the optimization process by declaring them to be false paths. Using this capability can help improve runtime and timing results.

In this exercise, you compile a different version of the STACK_FSM design. For the purposes of this exercise, you first identify the five paths with the worst timing (some or all of these paths might exhibit negative slack), and then you specify the four worst timing paths to be false paths, which you want Design Compiler to ignore. You perform an incremental compile to reoptimize the design. The recompiled design shows improved timing.

Note:

In a real design situation, the false paths you specify would be based on genuine design considerations. In this exercise, the false paths you specify are chosen only to demonstrate the effect of setting false paths.

Starting Design Vision for the False Path Exercise

You must exit Design Vision, if it is still running from a previous tutorial exercise. To quit the session, choose File > Exit and click OK in the warning box.

To start Design Vision in the dv_tutorial/risc_design-flse-path directory,

1. Open a UNIX shell on your terminal, if necessary.
2. Navigate to the dv_tutorial/risc_design-flse-path directory.
3. At the shell prompt, enter

   ```
   % design_vision
   ```

   The Design Vision window opens.

Reading, Constraining, and Compiling the STACK_FSM Design

To prepare the STACK_FSM design for the false path exercise, you repeat the procedures learned in Chapter 2, “Reading, Constraining, and Optimizing the Design.”

To read, constrain, and compile the STACK_FSM design,

1. Choose File > Read.

   The Read Designs dialog box appears.

   Choosing the Read menu command is equivalent to choosing the Analyze menu command followed by the Elaborate menu command.

2. Double click the source directory to open it, select STACK_FSM.vhd, and click Open.

   The name of the unmapped STACK_FSM design appears in the logical hierarchy view and in the Design list on the toolbar.

3. Choose File > Execute Script.

   The Execute Script File dialog box appears.

4. Double click the scripts directory to open it, select fsm_flse_path_constraints.tcl, and click Open.

   This is the command script file shown in Example A-2 on page A-5.

5. Choose Design > Check Design.

   The Check Design dialog box appears.

6. Make sure the “Display in detail” option and the “Current level and all sub-designs” option are selected; then click OK or Apply.

   No warnings or error messages appear in the console.

7. Choose Design > Compile Design.

   The Compile Design dialog box appears.

8. Make sure of the following settings:

   - The Mapping option “Map design” is selected.

     Make sure that both map effort and area effort are medium.

   - The Design rule option “Fix design rules and optimize mapping” is selected.

   - No other option is selected.

9. Click OK or Apply.

   The compilation takes a few minutes.

Equivalent dctcl Commands

```
read_file -format vhdl \

{./source/
STACK_FSM.vhd}

source ./dv_tutorial/risc_design-flse-path/scripts/
           fsm_flse_path_constraints.tcl
check_design
compile -map_effort medium -area_effort medium
```

Analyzing the Timing Results

Using the timing analysis methods described in Chapter 3, “Analyzing Timing and Area Results,” you identify the five worst slack paths produced by design optimization. Some or all five paths might have negative slack.

Note:

The following analysis quotes specific results for a particular set of worst timing paths.Your worst timing paths and their slack values might differ, but there will be no essential difference between your results and these. You will be able to apply the same procedure to your paths and achieve similar results.

To display the endpoint slack histogram for the STACK_FSM design,

1. Select STACK_FSM in the logical hierarchy view.
2. Choose Timing > Endpoint Slack.

   The Endpoint Slack dialog box appears.
3. Make sure Delay Type is set to max and Number Of Bins is set to 8.
4. Click OK or Apply.

   The endpoint slack histogram is displayed.
5. Place the pointer over the worst slack bin (red) and click.

   The endpoints-slack table to the right of the histogram shows the endpoint pin names with their negative slack values.
6. To view the startpoints for these paths, follow these steps:
   - Select the endpoints in the endpoints-slack table (Shift-click).
   - Keep the pointer over the selected paths, right-click, and choose “Select Timing Path to Selected Endpoints” from the pop-up menu.

     The number of selected paths appears in the status bar.

- Click Select > Selection List. The Selection List dialog box appears.

  The From Pin column in the Selection List table shows the startpoints of the selected endpoints. The endpoint paths are as follows (your paths and slack values might differ):

- PushEnbl to TOS_int_reg[0]/D
- PushEnbl to TOS_int_reg[2]/D
- PopEnbl to Crnt_Stack_reg[1]/S
- PushEnbl to TOS_int_reg[1]/D
- PopEnbl to Crnt_Stack_reg[0]/D

7. You use the `set_false_path` command to define the four worst paths as false paths. Enter the following four commands on the console command line:

```
set_false_path -from PushEnbl -to TOS_int_reg[0]/D
set_false_path -from PopEnbl -to Crnt_Stack_reg[0]/D
set_false_path -from PopEnbl -to Crnt_Stack_reg[1]/S
set_false_path -from PushEnbl -to TOS_int_reg[2]/D
```

Note:

Check your command input carefully. This method of entering commands is prone to typing errors. If you have entered a command that specifies an incorrect path, you can use the `reset_path` command (see the man page) to undo your error. Make sure, however, that you accurately enter the startpoint and endpoint names of the incorrect path when resetting the path. To avoid typing errors in repeated runs, put the correct commands in a script file.

Reoptimizing the Design and Comparing Timing Results

You do not need to do a full compile of the design. You use the incremental mapping option to recompile only the part of the design affected by the false path settings. Note that the constraints you applied to the design have not changed.

To reoptimize the STACK_FSM with false paths by using the incremental mapping option,

1. Select STACK_FSM in the logical hierarchy view.
2. Choose Design > Compile Design.

   The Compile Design dialog box appears.

3. Make sure of the following settings:

   - The Mapping option “Map design” is selected.

     Make sure that both map effort and area effort are high.

- The Design rule option “Fix design rules and optimize mapping” is selected.
- The Compile option “Incremental mapping” is selected.
- None of the other options are selected.

4. Click OK.
5. Repeat the timing analysis on the recompiled STACK_FSM design by generating an endpoint slack histogram and a timing report.

   The timing analysis shows that the timing results are generally improved.

   The timing report is similar to the one shown in Example 5-1:

Example 5-1 STACK_FSM Timing Report

```
****************************************
Report : timing
         -path full
         -delay max
         -max_paths 1
         -sort_by group
Design : STACK_FSM
Version: V-2003.12
Date   : Mon Oct 27 13:46:33 2003
****************************************

Operating Conditions:
Wire Load Model Mode: enclosed

  Startpoint: PushEnbl (input port clocked by my_clock)
  Endpoint: TOS_int_reg[0]
            (rising edge-triggered flip-flop clocked by my_clock)
  Path Group: my_clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  STACK_FSM          5KGATES               ssc_core

  Point                                    Incr       Path
  ----------------------------------------------------------
  clock my_clock (rise edge)               0.00       0.00
  clock network delay (ideal)              0.00       0.00
  input external delay                     2.00       2.00 r
  PopEnbl (in)                             0.00       2.00 r
  U166/Y (and3a15)                         0.18       2.18 r
  U165/Y (inv1a15)                         0.06       2.24 f
  U163/Y (mx2a15)                          0.18       2.42 f
  U161/Y (and2a15)                         0.13       2.55 f
  syn147/Y (oa1f6)                         0.16       2.70 r
  TOS_int_reg[0]/D (fdf1a1)                0.00       2.70 r
  data arrival time                                   2.70
```

```
  clock my_clock (rise edge)                 2.80       2.80
  clock network delay (ideal)                0.00       2.80
  TOS_int_reg[0]/CLK (fdf1a1)                0.00       2.80 r
  library setup time                        -0.11       2.69
  data required time                                    2.69
  ------------------------------------------------------------
  data required time                                    2.69
  data arrival time                                    -2.70
  ------------------------------------------------------------
  slack (VIOLATED)                                     -0.02
```

Equivalent dctcl Commands

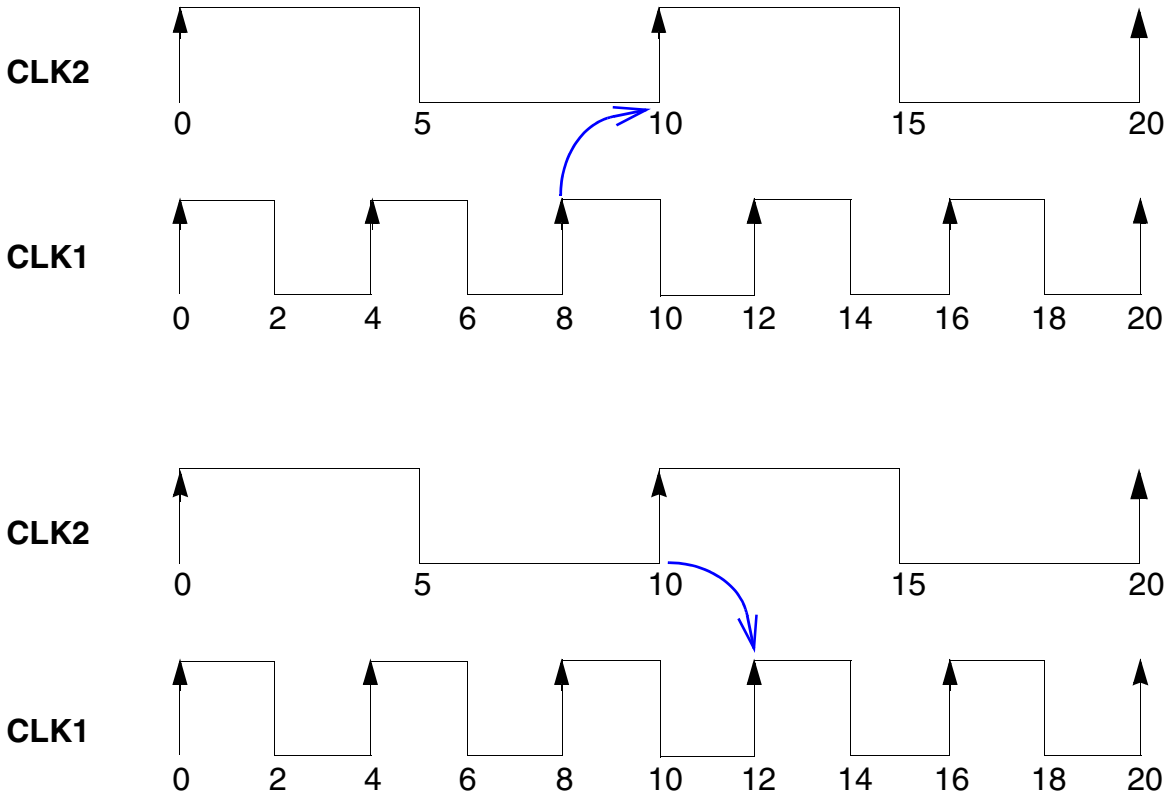
```
compile -map_effort high -area_effort high -incremental_mapping

report_timing -path full -delay max -nworst 1 \
      -max_paths 1 -significant_digits 2 -sort_by group
```

Optimizing a Design With Multiple Clocks

Designs often use several clocks running at different frequencies to control the transfer of data through combinational logic from one register to the next register. In such cases, timing paths under the control of two or more clocks can occur. That is, data can be launched from a register controlled by one clock and captured by a register controlled by a different clock. Because the clocks are running at different frequencies, the time available for transferring the data between registers depends on when the data is launched with respect to the two clock cycles.

Figure 5-1 shows an example with two clock waveforms. The first clock, Clk1, has a period of 4 ns, and the second clock, Clk2, has a period of 10 ns. (In this example, all registers in the design are assumed to launch and capture data on the rising edges of both clock pulses.) The clock waveforms have rising edge pulses at 0 ns. The rising edges do not coincide again until 20 ns. This 20-ns interval is referred to as the common base period. (The base period is an integer multiple of the two clock periods.) Note that between 0 and 20 ns, several values for the time interval between the rising edges of the two clocks are possible. This means that different paths can have very different launch-to-capture timing intervals to meet.

Figure 5-1 Clock Waveforms



In particular, data launched at 0, 4, and 8 ns from registers controlled by Clk1 can be captured at 10 ns by registers controlled by Clk2. Therefore, three different arrival times are possible: data launched at 0 ns has 10 ns to reach the second register, data launched at 4 ns has 6 ns, and data launched at 8 ns has 2 ns.

Clearly, the 2-ns interval presents the worst case for timing optimization. Design Compiler works to meet this timing constraint first.

Similarly, the worst-case time interval for data launched under the control of Clk2 and captured under the control of Clk1 is 2 ns and occurs from 10 ns to 12 ns.

In this exercise, you examine the effects of multiple clocks on timing optimization. You define the two clocks, as shown in Figure 5-1, for the STACK_FSM design, set the timing constraints, optimize the design, and then analyze the timing results to see how Design Compiler handles the worst-case timing paths.

Starting Design Vision for the Multiple Clock Exercise

You must exit Design Vision, if you still have it running from a previous tutorial exercise. To quit the session, choose File > Exit and click OK in the warning box.

To start Design Vision in the dv_tutorial/risc_design-mult-clk directory,

1. Open a UNIX shell on your terminal, if necessary.
2. Navigate to the dv_tutorial/risc_design-mult-clk directory.
3. At the shell prompt, enter

   ```
   % design_vision
   ```

   The Design Vision window opens.

Reading the STACK_FSM Into Memory

You begin this exercise by reading the STACK_FSM design into memory. Instead of using the `analyze` and `elaborate` commands, you use the `read` command, which achieves the results of both commands.

To read the STACK_FSM design,

1. Choose File > Read.

   The Read Designs dialog box appears.
2. Double click the source directory to open it, select STACK_FSM.vhd, and click Open.

   The name of the unmapped STACK_FSM design appears in the logical hierarchy view and in the Design list on the toolbar.

You can now define the clocks and apply the timing constraints.

Equivalent dctcl Commands

```
read_file -format vhdl {./source/STACK_FSM.vhd}
```

Defining Multiple Clocks

The STACK_FSM design has two clock input ports, Clk1 and Clk2. You create two clocks with different frequencies, assigning one clock to the Clk1 port and the other clock to the Clk2 port. Each clock defines a clock domain. Timing paths belong to a particular clock domain, depending on which clock controls the path endpoint register.

To define two clocks for the STACK_FSM design,

1. Select STACK_FSM in the logical hierarchy view.
2. Click the Create Symbol View button on the toolbar (or select Schematic > New Symbol View).

   The symbol view of the STACK_FSM design is displayed. There are two clock input ports, Clk1 and Clk2.
3. Click the Clk1 port in the symbol view.
4. Choose Attributes > Specify Clock.

   The Specify Clock dialog box appears.
5. Enter my_clock1 in the Clock name box.
6. Make sure Clk1 appears in the Port name box.

   You get the correct port name by clicking the appropriate clock port in the symbol view of the design.
7. The default clock waveform has a half-duty cycle pulse with a rising edge at 0 and a falling edge at T/2, where T is the clock period. Do the following to define the clock attribute on port Clk1:
   - Enter the value 4 in the Period box.
   - In the Edge-Value table, enter the value 0 in the Rising value box, and press Return. In a similar manner, enter the value 2 in the Falling value box.

     With only a single edge value specified, the first rising edge occurs at the time specified, and the falling edge occurs at the time (period_value/2) later.
   - Click OK.

     The clock waveform is now a square-wave clock pulse with a period of 4 ns.
8. Repeat steps 3 to 7 to define the second clock attribute on port Clk2. Make sure of the following:
   - Port Clk2 is selected in the symbol view.
   - Clock name is my_clock2.

- Port name is Clk2.
- Period is 10.
- Rising value is 0 and Falling value is 5.

9. Click OK.

Equivalent dctcl Commands

```
create_clock -name "my_clock1" -period 4 -waveform { 0 2 }  { Clk1 }

create_clock -name "my_clock2" -period 10 -waveform { 0 5 } { Clk2 }
```

Setting Input and Output Delays on the STACK_FSM Ports

Input and output delays for the design are defined with respect to the clocks of the design. In this exercise, you set maximum delays relative to my_clock1 and my_clock2. You set the same input delay of 2 ns on all input ports with respect to the two clocks. Similarly, you set the same output delay of 0.5 ns on all output ports.

Enter the following four commands on the console command line:

```
set_input_delay 2.0 -max -clock my_clock1 \
[remove_from_collection [all_inputs] [get_ports Clk1]]

set_input_delay 2.0 -max -clock my_clock2 \
[remove_from_collection [all_inputs] [get_ports Clk2]]

set_output_delay 0.5 -max -clock my_clock1 [all_outputs]

set_output_delay 0.5 -max -clock my_clock2 [all_outputs]
```

Note:

As before, check your command input carefully. If you have entered an incorrect command, you can correct your mistake by entering the correct command. When you must enter commands with complicated syntax, it's good practice to put the commands in a script to minimize chance errors and speed up multiple runs.

In this exercise, instead of entering the commands, you can use the constraints script fsm_mult_clk_constraints.tcl shown in Example A-3 on page A-6.

Optimizing the STACK_FSM Design

With the clocks and maximum input and output delays specified, you can now optimize the design for two clocks.

To check and compile the STACK_FSM design,

1. Choose Design > Check Design.

   The Check Design dialog box appears.
2. Make sure the “Display in detail” option and “Current level and all sub-designs” option are selected.
3. Click OK.
4. Read the contents of the log view in the console.

   You might see a number of warning messages regarding nets that have multiple drivers. For the purposes of this tutorial, these warning messages are not important, and you can ignore them.
5. Choose Design > Compile Design.

   The Compile Design dialog box appears.
6. Make sure of the following settings:
   - The Mapping option “Map design” is selected.

     Make sure that both map effort and area effort are medium.
   - The Design rule option “Fix design rules and optimize mapping” is selected.
   - No other option is selected.
7. Click OK.

   The compilation takes a few minutes.

Equivalent dctcl Commands

```
check_design
compile -map_effort medium -area_effort medium
```

Analyzing the Timing Results

In this part of the exercise, you use several methods to analyze the timing results of the optimized STACK_FSM design. These methods make use of

- Critical path schematics
- Endpoint slack histograms
- Path timing reports

Note:
Although your results might differ slightly from the results documented here, the following analysis applies.

Finding the Critical Paths

You can quickly display the critical path or paths in a design schematic.

To display critical paths in the schematic view,

1. Display the STACK_FSM design schematic by choosing, Schematic > New Design Schematic View.

   Make sure the schematic view is the active view.

2. Select the critical paths by choosing, Select > Paths From/Through/To, and then clicking OK in the Select Paths dialog box.

   The default options are set to select the path or paths with the worst negative slack in the design.

3. Highlight the selected paths by choosing, Highlight > Selected.

4. Use the magnification tools and the pointer to identify the startpoints and endpoints of the critical path.

Analyzing the Endpoint Slack Histograms, Path Schematics, and Path Timing Reports

To get a quick overview of the optimization results, generate an endpoint slack histogram for the STACK_FSM design.

Note:
The following analysis quotes specific results for a particular set of worst timing paths.Your worst timing paths and their slack values might differ, but there will be no essential difference between your results and these. You will be able to apply the same procedure to your paths and achieve similar results.

The histogram shows two paths with a negative slack of -0.219127 for endpoints Crnt_Stack_reg[0]/E and Crnt_Stack_reg[1]/E. Right-click and use the pop-up menu to create the path schematic for the two endpoints. You can see that the startpoint for both paths is the Reset input port.

Choose Timing > Report Timing to obtain a path timing report for each selected endpoint with negative slack. The path timing reports show that negative slack occurs in the 10- to 12-ns time frame, with the Reset port controlled by my_clock2 and the endpoints Crnt_Stack_reg[0]/E and Crnt_Stack_reg[1]/E controlled by my_clock1. You would expect this result from your analysis of Figure 5-1 on page 5-8.

Note:

These two paths are ignored for the remainder of this exercise for the following reasons: They contain no intervening combinational logic and are not critical to optimization results. They exhibit negative slack in the 10- to 12-ns time frame only because (1) they are constrained incorrectly by an external input delay of 2 ns and (2) they end at registers with setup times of 0.22 ns.

Generate the endpoints-slack table for the bin with the smallest positive slack values. Select the two endpoints TOS_int_reg[0]/D and TOS_int_reg[1]/D. Create path schematics and path timing reports for each of these endpoints by using the pop-up menu.

The path schematics show that these paths are virtually identical. They have the same startpoint, Crnt_Stack_reg[1]/CLK. The two paths differ only in the endpoint registers, which reference the same design.

In addition, the two path timing reports are almost identical. The path timing reports show that the worst positive slack occurs in the 8- to 10-ns time frame with my_clock1 controlling data launch and my_clock2 controlling data capture. This is the other worst-case path that you would expect from your analysis of Figure 5-1 on page 5-8.

The information already presented in this section can be found in a timing path report.

To display a timing path report for the five worst paths,

1. Choose Timing > Report Timing.
2. Enter 5 in the Max paths per group box of the Timing Report dialog box.

The report, similar to the one shown in Example 5-2, appears in a new report view of the console.

Example 5-2 Multiple-Clock Timing Report

```
****************************************
Report : timing
         -path full
         -delay max
         -max_paths 5
         -sort_by group
Design : STACK_FSM
Version: V-2003.12
Date   : Mon Oct 27 16:58:04 2003
****************************************

Operating Conditions:
Wire Load Model Mode: enclosed

  Startpoint: Reset (input port clocked by my_clock2)
  Endpoint: Crnt_Stack_reg[0]
            (rising edge-triggered flip-flop clocked by my_clock1)
  Path Group: my_clock1
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  STACK_FSM          5KGATES               ssc_core

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock my_clock2 (rise edge)             10.00      10.00
  clock network delay (ideal)              0.00      10.00
  input external delay                     2.00      12.00 f
  Reset (in)                               0.00      12.00 f
  Crnt_Stack_reg[0]/E (fdef1a15)           0.00      12.00 f
  data arrival time                                  12.00

  clock my_clock1 (rise edge)             12.00      12.00
  clock network delay (ideal)              0.00      12.00
  Crnt_Stack_reg[0]/CLK (fdef1a15)         0.00      12.00 r
  library setup time                      -0.22      11.78
  data required time                                 11.78
  -----------------------------------------------------------
  data required time                                 11.78
  data arrival time                                 -12.00
  -----------------------------------------------------------
  slack (VIOLATED)                                   -0.22

  Startpoint: Reset (input port clocked by my_clock2)
  Endpoint: Crnt_Stack_reg[1]
            (rising edge-triggered flip-flop clocked by my_clock1)
  Path Group: my_clock1
  Path Type: max
```

```
Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
STACK_FSM          5KGATES               ssc_core

Point                                    Incr       Path
-----------------------------------------------------------

clock my_clock2 (rise edge)             10.00      10.00
clock network delay (ideal)              0.00      10.00
input external delay                     2.00      12.00 f
Reset (in)                               0.00      12.00 f
Crnt_Stack_reg[1]/E (fdef1a15)           0.00      12.00 f
data arrival time                                  12.00

clock my_clock1 (rise edge)             12.00      12.00
clock network delay (ideal)              0.00      12.00
Crnt_Stack_reg[1]/CLK (fdef1a15)         0.00      12.00 r
library setup time                      -0.22      11.78
data required time                                 11.78
-----------------------------------------------------------
data required time                                 11.78
data arrival time                                 -12.00
-----------------------------------------------------------
slack (VIOLATED)                                   -0.22

Startpoint: Crnt_Stack_reg[0]
            (rising edge-triggered flip-flop clocked by my_clock1)
Endpoint: TOS_int_reg[0]
          (rising edge-triggered flip-flop clocked by my_clock2)
Path Group: my_clock2
Path Type: max

Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
STACK_FSM          5KGATES               ssc_core

Point                                    Incr       Path
-----------------------------------------------------------
clock my_clock1 (rise edge)              8.00       8.00
clock network delay (ideal)              0.00       8.00
Crnt_Stack_reg[0]/CLK (fdef1a15)         0.00       8.00 r
Crnt_Stack_reg[0]/Q (fdef1a15)           0.49       8.49 f
U152/Y (and2a15)                         0.13       8.63 f
U153/Y (or2a15)                          0.23       8.86 f
U148/Y (inv1a27)                         0.06       8.92 r
U144/Y (or2a15)                          0.14       9.06 r
U147/Y (or2a15)                          0.14       9.20 r
U146/Y (inv1a15)                         0.07       9.27 f
U150/Y (ao1a15)                          0.26       9.54 f
U129/Y (or3a15)                          0.30       9.84 f
TOS_int_reg[0]/D (fdf1a1)                0.00       9.84 f
data arrival time                                   9.84
```

```
clock my_clock2 (rise edge)              10.00      10.00
clock network delay (ideal)               0.00      10.00
TOS_int_reg[0]/CLK (fdf1a1)               0.00      10.00 r
library setup time                       -0.15       9.85
data required time                                   9.85
-----------------------------------------------------------
data required time                                   9.85
data arrival time                                   -9.84
-----------------------------------------------------------
slack (MET)                                          0.01

Startpoint: Crnt_Stack_reg[0]
            (rising edge-triggered flip-flop clocked by my_clock1)
Endpoint: TOS_int_reg[1]
          (rising edge-triggered flip-flop clocked by my_clock2)
Path Group: my_clock2
Path Type: max

Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
STACK_FSM          5KGATES               ssc_core

Point                                    Incr       Path
-----------------------------------------------------------
clock my_clock1 (rise edge)              8.00       8.00
clock network delay (ideal)              0.00       8.00
Crnt_Stack_reg[0]/CLK (fdef1a15)         0.00       8.00 r
Crnt_Stack_reg[0]/Q (fdef1a15)           0.49       8.49 f
U152/Y (and2a15)                         0.13       8.63 f
U153/Y (or2a15)                          0.23       8.86 f
U148/Y (inv1a27)                         0.06       8.92 r
U144/Y (or2a15)                          0.14       9.06 r
U147/Y (or2a15)                          0.14       9.20 r
U146/Y (inv1a15)                         0.07       9.27 f
U150/Y (ao1a15)                          0.26       9.54 f
U149/Y (or3a15)                          0.30       9.84 f
TOS_int_reg[1]/D (fdf1a1)                0.00       9.84 f
data arrival time                                   9.84

clock my_clock2 (rise edge)              10.00      10.00
clock network delay (ideal)               0.00      10.00
TOS_int_reg[1]/CLK (fdf1a1)               0.00      10.00 r
library setup time                       -0.15       9.85
data required time                                   9.85
-----------------------------------------------------------
data required time                                   9.85
data arrival time                                   -9.84
-----------------------------------------------------------
slack (MET)                                          0.01
```

```
Startpoint: Crnt_Stack_reg[0]
            (rising edge-triggered flip-flop clocked by my_clock1)
Endpoint: Crnt_Stack_reg2[0]
          (rising edge-triggered flip-flop clocked by my_clock2)
Path Group: my_clock2
Path Type: max

Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
STACK_FSM          5KGATES               ssc_core

Point                                    Incr       Path
-----------------------------------------------------------
clock my_clock1 (rise edge)              8.00       8.00
clock network delay (ideal)              0.00       8.00
Crnt_Stack_reg[0]/CLK (fdef1a15)         0.00       8.00 r
Crnt_Stack_reg[0]/Q (fdef1a15)           0.50       8.50 r
U130/Y (and2a6)                          0.19       8.69 r
U151/Y (ao1a1)                           0.83       9.52 r
U156/Y (or3a6)                           0.31       9.83 r
Crnt_Stack_reg2[0]/D (fdef1a3)           0.00       9.83 r
data arrival time                                   9.83

clock my_clock2 (rise edge)             10.00      10.00
clock network delay (ideal)              0.00      10.00
Crnt_Stack_reg2[0]/CLK (fdef1a3)         0.00      10.00 r
library setup time                      -0.15       9.85
data required time                                  9.85
-----------------------------------------------------------
data required time                                  9.85
data arrival time                                  -9.83
-----------------------------------------------------------
slack (MET)                                         0.02

Startpoint: Crnt_Stack_reg[0]
            (rising edge-triggered flip-flop clocked by my_clock1)
Endpoint: Crnt_Stack_reg2[1]
          (rising edge-triggered flip-flop clocked by my_clock2)
Path Group: my_clock2
Path Type: max

Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
STACK_FSM          5KGATES               ssc_core

Point                                    Incr       Path
-----------------------------------------------------------
clock my_clock1 (rise edge)              8.00       8.00
clock network delay (ideal)              0.00       8.00
Crnt_Stack_reg[0]/CLK (fdef1a15)         0.00       8.00 r
Crnt_Stack_reg[0]/Q (fdef1a15)           0.50       8.50 r
U130/Y (and2a6)                          0.19       8.69 r
U151/Y (ao1a1)                           0.83       9.52 r
```

```
U157/Y (or3a6)                           0.31       9.83 r
Crnt_Stack_reg2[1]/D (fdef1a3)           0.00       9.83 r
data arrival time                                   9.83

clock my_clock2 (rise edge)             10.00      10.00
clock network delay (ideal)              0.00      10.00
Crnt_Stack_reg2[1]/CLK (fdef1a3)         0.00      10.00 r
library setup time                      -0.15       9.85
data required time                                  9.85
-----------------------------------------------------------
data required time                                  9.85
data arrival time                                  -9.83
-----------------------------------------------------------
slack (MET)                                         0.02

Startpoint: Crnt_Stack_reg[0]
            (rising edge-triggered flip-flop clocked by my_clock1)
Endpoint: Crnt_Stack_reg2[0]
          (rising edge-triggered flip-flop clocked by my_clock2)
Path Group: my_clock2
Path Type: max

Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
STACK_FSM          5KGATES               ssc_core

Point                                    Incr       Path
-----------------------------------------------------------
clock my_clock1 (rise edge)              8.00       8.00
clock network delay (ideal)              0.00       8.00
Crnt_Stack_reg[0]/CLK (fdef1a15)         0.00       8.00 r
Crnt_Stack_reg[0]/Q (fdef1a15)           0.50       8.50 r
U130/Y (and2a6)                          0.19       8.69 r
U139/Y (or2a1)                           0.77       9.46 r
Crnt_Stack_reg2[0]/E (fdef1a3)           0.00       9.46 r
data arrival time                                   9.46

clock my_clock2 (rise edge)             10.00      10.00
clock network delay (ideal)              0.00      10.00
Crnt_Stack_reg2[0]/CLK (fdef1a3)         0.00      10.00 r
library setup time                      -0.33       9.67
data required time                                  9.67
-----------------------------------------------------------
data required time                                  9.67
data arrival time                                  -9.46
-----------------------------------------------------------
slack (MET)                                         0.20
```

Study this report carefully and verify that it contains all the information of the previous analysis.

Equivalent dctcl Commands

```
report_timing -to { Crnt_Stack_reg[0]/E } -path full \
      -delay max -nworst 1 -max_paths 1 \
      -significant_digits 2 -sort_by group

report_timing -to { Crnt_Stack_reg[1]/E } -path full \
      -delay max -nworst 1 -max_paths 1 \
      -significant_digits 2 -sort_by group

report_timing -to { TOS_int_reg[0]/D0 } -path full \
      -delay max -nworst 1 -max_paths 1 \
      -significant_digits 2 -sort_by group

report_timing -to { TOS_int_reg[1]/D0 } -path full \
      -delay max -nworst 1 -max_paths 1\
      -significant_digits 2 -sort_by group

report_timing -delay max -nworst 1 -max_paths 5 \
      -significant_digits 2 -sort_by group
```

A

Design Constraints

This appendix provides background information on design constraints. It is intended as an overview only. No tutorial exercises are included. However, it does include copies of the constraint files that you use in the tutorial exercises. For more information on constraints, see the Design Compiler documentation suite and the appropriate man pages.

You should read this appendix before applying constraints to the RISC_CORE and STACK_FSM designs as instructed in the tutorial exercises.

This appendix includes the following sections:

Design Constraints Overview

Design constraints consist of the design environment definition, design rules, and optimization constraints.

Design Environment Definition

The design environment is defined by the following:

- Operating conditions

  These conditions include operating temperature, supply voltage, and manufacturing process.

- Wire load models

  These models, which include top, enclosed, and segmented modes, estimate the effect of wire length on design performance.

- System interface characteristics of the I/O ports

  These characteristics include input drives, input and output loads, and fanout loads on the design's ports.

The environment parameters, models, drives, and loads are provided by the technology library. Except for setting the drive on an input port, you define the design environment by choosing library environment variables. The input drive is usually set according to a library drive cell value, but you can also set it directly.

Design Rules

Design rules are supplied in the technology library you use and include the following constraints:

- Transition time
- Fanout load
- Output pin capacitance
- Cell degradation
- Connection class

For the design to function properly, Design Compiler must meet the library constraints (implicit constraints) when it compiles the design. You can define tighter design rule constraints (explicit constraints), but you cannot relax the constraint values defined in the technology library.

Note:
The tutorial exercises do not use the cell degradation and connection class constraints.

Optimization Constraints

Optimization constraints specify the design *goals* you want Design Compiler to meet during optimization and include the following constraints:

- Timing, which comprises
  - I/O timing requirements
  - Combinational path delays
  - Timing exceptions
- Maximum area

Design Compiler attempts to meet the goals you set, but it does not violate the design rules in order to meet optimization goals.

Methods for Specifying Constraints

You can specify the environment, design rule, and optimization constraints in the following ways:

- Use the Attributes menu and submenus of Design Vision to define clocks, the operating environment, design rules, and optimization goals. With this method, you do not have to know the complex syntax of the many commands used to define the various constraints.
- Enter the specific commands that define the constraints on the console command line. With this method, you have to know the syntax of the commands.
- Execute a script file of commands that define the constraints. Constraint scripts are text files that can be developed several ways. An easy way to make constraints files for designs that have characteristics and design goals similar to your current design is to cut the appropriate commands from the view_command.log file of an optimization session, paste them into a text file, and modify them as needed.

  Also, after you are familiar with the commands and their syntax, you can type them directly into the script file. You can combine and edit script files to make new script files.

Constraint Script Files

In the main tutorial exercise of Chapters 2, 3, and 4, the constraints are defined for you in the script file top_level.tcl (Example A-1 on page A-5). In the false path and multiple clock tutorial exercises of Chapter 5, the constraints are defined in the script files fsm_flse_path_constraints.tcl (Example A-2 on page A-5) and fsm_mult_clk_constraints.tcl (Example A-3 on page A-6).

Before applying each script, study the individual commands listed in the example. To learn more about any command in the script files, you can read the man page for the command by typing the following on the console command line:

```
man command_name
```

For example,

```
man create_clock
```

displays the man page information on the `create_clock` command in the console. Accessing the man page is a quick way to get important information about a command.

The constraint files consist of four sections:

- Creation of user-defined variables

  In this section, both the variables referenced in the commands that follow in the file and their values are defined. The variable names are arbitrary; however, the user-defined names chosen for these variables in this script suggest the type of constraint to be applied to the design. The use of user-defined variables makes it easy to change constraint values. The commands in which the variables appear do not have to be retyped every time you change the value of a variable.

- Definition of top-level budget

  In this section of the script file, the clock is defined, including the clock period and clock skew, and the clock ports are identified; also, the input and output delays for all inputs and outputs are specified. A `set_dont_touch_network` attribute is placed on the clock network to prevent Design Compiler from adding buffers to the clock network during optimization.

- Area constraint

  In this section, a single command sets the maximum area optimization goal.

- Operating environment

  In this section, the operating conditions, wire load model, driving cell, and maximum output load on all outputs are defined.

Example A-1 Top-Level Constraints File top-level.tcl

```
# Create User-Defined Variables
set CLK_PORT [get_ports Clk]
set CLK_PERIOD 4.00
set CLK_SKEW 0.14
set WC_OP_CONDS typ_0_1.98
set WIRELOAD_MODEL 10KGATES
set DRIVE_CELL buf1a6
set DRIVE_PIN {Y}
set MAX_OUTPUT_LOAD [load_of ssc_core/buf1a2/A]
set INPUT_DELAY 2.0
set OUTPUT_DELAY 0.5
set MAX_AREA 380000

# Define Top-Level Budget
create_clock -period $CLK_PERIOD -name my_clock $CLK_PORT
set_dont_touch_network my_clock
set_clock_uncertainty $CLK_SKEW [get_clocks my_clock]

set_input_delay $INPUT_DELAY -max -clock my_clock \
     [remove_from_collection [all_inputs] $CLK_PORT]
set_output_delay $OUTPUT_DELAY -max -clock my_clock [all_outputs]

# Set Area Constraint
set_max_area $MAX_AREA

# Set Operating Environment
set_operating_conditions -max $WC_OP_CONDS
set_wire_load_model -name $WIRELOAD_MODEL
set_driving_cell -lib_cell $DRIVE_CELL -pin $DRIVE_PIN \
   [remove_from_collection [all_inputs] $CLK_PORT]
set_load $MAX_OUTPUT_LOAD [all_outputs]
```

Example A-2 False Path Constraints File fsm_flse_path_constraints.tcl

```
create_clock -period 2.8 -name my_clock [get_ports Clk]

set_input_delay 2.0 -max -clock my_clock \
     [remove_from_collection [all_inputs] [get_ports Clk]]

set_false_path -from Reset -to STACK_FULL
set_output_delay 0.5 -max -clock my_clock [all_outputs]
```

Example A-3 Multiple Clocks Constraints File fsm_mult_clk_constraints.tcl

```
create_clock -period 4.0 -name my_clock1 [get_ports Clk1]
create_clock -period 10.0 -name my_clock2 [get_ports Clk2]

set_input_delay 2.0 -max -clock my_clock1 \
     [remove_from_collection [all_inputs] [get_ports Clk1]]

set_input_delay 2.0 -max -clock my_clock2 \
     [remove_from_collection [all_inputs] [get_ports Clk2]]

set_output_delay 0.5 -max -clock my_clock1 [all_outputs]
set_output_delay 0.5 -max -clock my_clock2 [all_outputs]
```

Index

D

E

F

G

H

I

L

M